UNIVERSIDAD POLITECNICA DE VALENCIA

UNIVERSIDAD POLITÉCNICA DE VALENCIA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

MIGRACIÓN DE LA APLICACIÓN DE ARCHIVO DE EXPEDIENTES JUDICIALES

ALBERTO MARTÍNEZ HERRERO

Tesis para optar al grado de
Máster en Ingeniería del Software Métodos Formales y Sistemas de Información.

Profesor Supervisor:
VICENTE PELECHANO FERRAGUD
Tutor en la empresa:
SERGIO GUILLEN COMPANY

Valencia, Diciembre de 2008.

ÍNDICE DE FIGURAS

ABREVIATURAS.

Abreviación	Descripción
J2EE	Java 2 Enterprise Edition
EJB	Enterprise Java Bean
JNDI	Java Naming and Directory Interface
JAR	Java ARchive
JSP	JavaServer Pages
XML	eXtensible Markup Language
HTML	HyperText MarkUp Language
SQL	Structured Query Language
PDF	Portable Document Format
MDA	Model Driven Architecture
OMG	Object Management Group
MDE	Model Driven Engineering
XMI	XML Metadata Interchange
QVT	Query View Transformation
XSL	eXtensible Stylesheet Language
SGBD	Sistema de Gestión de Base de Datos
RUP	Rational Unified Process

1 INTRODUCCIÓN

En el capítulo de introducción, se pretende que el lector pueda hacerse una idea clara de lo que va a encontrar en el resto del documento.

Para alcanzar este objetivo comenzaremos presentando un breve resumen de los contenidos del documento y se situará su realización en un contexto concreto. Seguidamente se enumerarán los objetivos marcados y será justificada la necesidad del cliente de utilizar la herramienta de gestión que nos ocupa.

Como conclusión del capítulo se detallará cual será la estructura del resto del documento.

1.1 Resumen

Este documento contiene una tesis de orientación puramente profesional, en la que se detalla el desarrollo de un proyecto software empresarial, en el contexto de la multinacional de Tecnologías de la Información número 1 en España y una de las principales de Europa y Latinoamérica.

El proyecto, consiste en la migración de la aplicación para la gestión de expedientes judiciales, utilizada en la Ciudad de la Justicia de Valencia, para adaptarla a la versión 1.4.1 del framework de desarrollo utilizado por el Ministerio de Justicia. Para, de este modo, integrarla con la aplicación Minerva y su sistema de información.

En ella se aplicarán parte de las tecnologías que forman parte de los contenidos del máster, como puede ser J2EE o el SGBD Oracle.

Además se pretende poner de manifiesto que si se hubiesen aplicado otras tecnologías incluidas en los contenidos del máster, como la arquitectura MDA, podrían haberse reducido los costes del proyecto.

Para ello se detallarán los problemas y dificultades encontrados durante el desarrollo del proyecto y se propondrán soluciones alternativas que los habrían evitado, o al menos, reducido en gran medida.

1.2 Contextualización

Esta tesis ha sido realizada en el marco de la empresa Indra Sistemas, empresa en la que existe un convenio en la actualidad con la Universidad Politécnica de Valencia.

Indra Sistemas S.A. es una compañía multinacional de capital español, del sector de las Tecnologías de la Información, sistemas de defensa, sistemas electrónicos, transporte y consultoría tecnológica. A finales de 2007 tenía una plantilla de más de 20.000 empleados tras la absorción, realizada ese mismo año, de Soluziona y Azertia, entre otras.

El proyecto ha sido desarrollado en el centro que tiene la compañía en la calle Isabel la Católica nº 8 en Valencia, dentro del departamento de Sanidad y Administraciones Públicas.

Esta aplicación es un componente dentro del proyecto del proyecto Minerva con el que se deberá realizar la integración. Está basada en la aplicación con el mismo nombre que se está utilizando en la actualidad en la Generalitat Valenciana.

El proyecto Minerva es una aplicación del Ministerio de Justicia que será utilizada en la comunidad de Murcia, la finalidad de esta aplicación es registrar informáticamente toda la información relativa a los asuntos judiciales.

A continuación vemos una captura en la que podemos apreciar la información recogida en un registro.

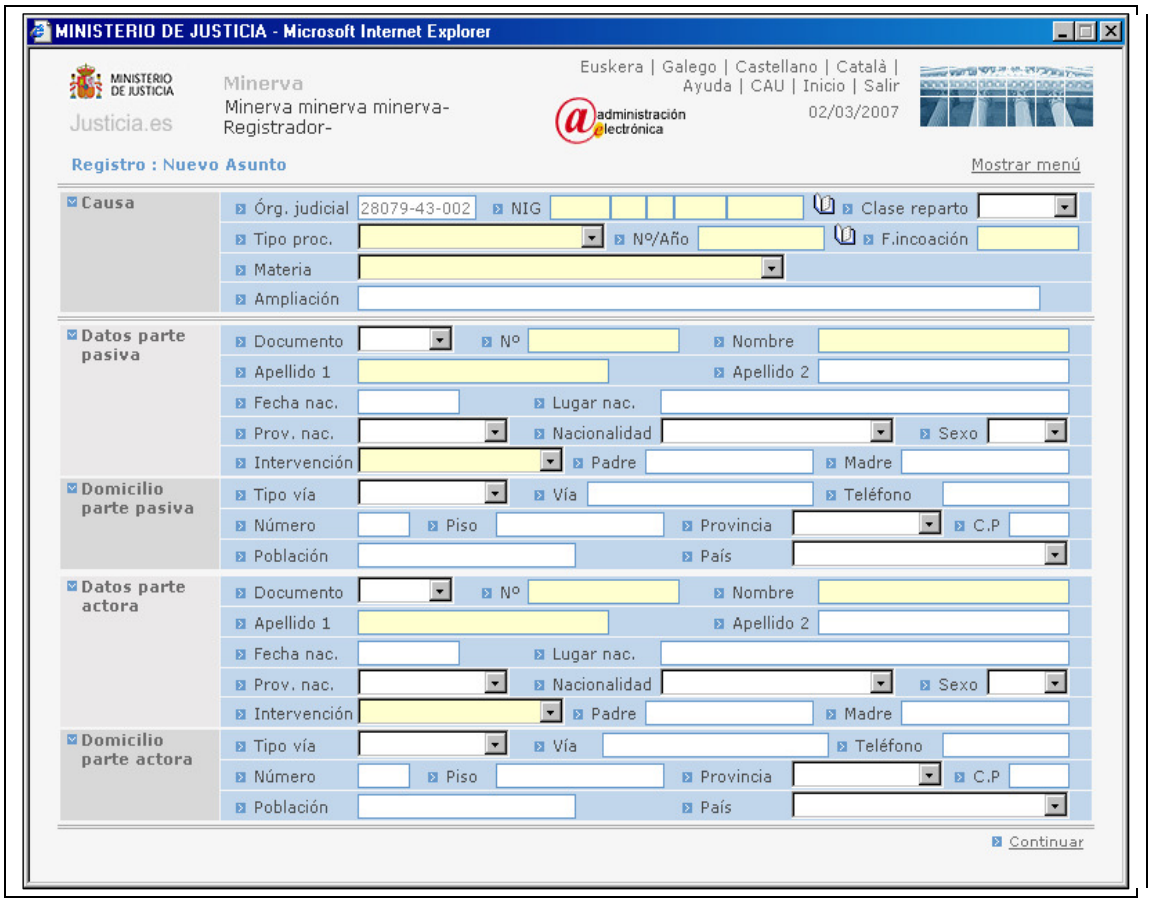

Figura 1. Registro de expedientes en Minerva

Además del registro de asuntos la aplicación proporciona diferentes utilidades de consulta, informes y estadísticas.

El mecanismo de comunicación de las dos aplicaciones es muy sencillo. Al utilizar ambas aplicaciones el mismo esquema de base de datos se han creado una serie de tablas que sirven de bandejas de entrada y salida.

En la bandeja de entrada el archivo recibe tanto nuevos expedientes de los juzgados como solicitudes de préstamo.

En la bandeja de salida los juzgados pueden recoger aquellos expedientes para los que el Archivo Judicial de Gestión ha aceptado la solicitud de préstamo.

1.3 Objetivos

Con la realización de este trabajo se pretenden alcanzar una serie de objetivos que se enumeran a continuación.

1.3.1 Comunes con la empresa

- Migración de la aplicación de Archivo de Expedientes Judiciales a la versión 1.4.1 Del Framework del ministerio de justicia.
- Utilización del modelo de datos propio de Minerva.
- Realizar algunas modificaciones en el diseño de la aplicación.
- Adaptar la apariencia visual a la guía impuesta por el Ministerio.

1.3.2 Propios del trabajo

Una vez realizada la migración de la aplicación, analizar los problemas encontrados y proponer alternativas futuras que reduzcan los costes, mediante la aplicación de técnicas semiautomáticas estudiadas a lo largo del máster.

1.4 Justificación

En las siguientes líneas queda justificada la necesidad del desarrollo y la utilización de una aplicación de este tipo, como solución a los problemas que están presentes en la actualidad en los archivos judiciales.

1.4.1 Real Decreto 937/2003

En julio de 2003 se ha publicado un Real Decreto de Modernización de los Archivos Judiciales que plantea la necesidad de fijar un sistema de gestión y custodia de la documentación judicial Real Decreto 937/2003, de 18 de julio, de modernización de los archivos judiciales.

Preámbulo del real decreto:

Este real decreto [5] es consecuencia de las numerosas reformas legales que se están llevando a cabo para la ***mejora y modernización de la justicia****. Los españoles, comenzando por los diferentes operadores jurídicos, demandan inequívocamente un esfuerzo profundo en este sentido. El Pacto de Estado para la reforma de la justicia prevé su mejora y actualización de acuerdo con las necesidades sociales. Dicha mejora se debe sustentar en la* ***modernización de la oficina judicial*** *y en la dotación de una serie de medios, a través de un plan de infraestructuras que supla determinadas carencias. Ambos puntos convergen en la conveniencia de crear una oficina judicial ágil, rápida y con una correcta atención al ciudadano.* ***Una oficina judicial saturada de expedientes difíciles de ubicar o archivar, con la consiguiente dificultad para encontrarlos en ocasiones produce en la sociedad la imagen de una justicia lenta e ineficaz****.*

Es necesario fijar un sistema de gestión y custodia de la documentación judicial por el que se descongestionen los diferentes juzgados y tribunales*, otorgando a cada uno de ellos su propio archivo con el que clasificar y custodiar todos aquellos expedientes que se encuentren en tramitación. Por el contrario, los que no están pendientes de tramitación se podrán enviar a los archivos territoriales o centrales o, en su caso, a la Junta de Expurgo, evitando así que ocupen un espacio innecesario. Esto exige una* ***actualización y unificación de la normativa que regule el expurgo de los archivos de los juzgados y tribunales****, así como el establecimiento de criterios que garanticen la más idónea conservación de cuantos documentos pudieran tener valor cultural, histórico, jurídico o administrativo, pues no hay que olvidar que la documentación que produce la Administración de Justicia constituye parte integrante del Patrimonio Documental y Bibliográfico.*

Este real decreto pretende la regulación unitaria y conjunta de esta materia, adecuándola a la realidad social y jurídica actual, consiguiendo un equilibrio entre la tradicional técnica archivística y el desarrollo creciente de las nuevas tecnologías*.*

Por otra parte, la Constitución española, en su artículo 46,obliga a los poderes públicos a ***garantizar y promover el enriquecimiento del patrimonio histórico y cultural*** *de los pueblos de España, con independencia de su régimen jurídico y su titularidad, a la vez que reconoce a los ciudadanos, en el artículo 105.b),el* ***derecho de acceso a los archivos y registros*** *administrativos, salvo en lo que afecte a la seguridad y defensa del Estado, la averiguación de los delitos y la intimidad de la persona.*

1.4.2 Situación general de los archivos judiciales en España

Principales problemas y necesidades (con carácter general) **medios insuficientes, inadecuados y desbordados para organizar, conservar y gestionar el enorme y creciente volumen de documentos que se** y que se manifiesta en:

- Gran acumulación de documentos en oficinas judiciales y saturación de sus instalaciones de archivo.
- Invasión de los expedientes de los pasillos, servicios, y otros ámbitos de atención al público, que comprometen la confidencialidad y seguridad de los expedientes, y generan una deplorable imagen en la opinión pública y en los medios de comunicación.
- Adopción de medidas de urgencia que originan expurgos indiscriminados
- Ausencia de una estructura archivística y falta de planificación y política archivística.
- Riesgo constante de extravío, pérdida de control, seguridad y acceso a los expedientes judiciales, etc.
- Caos y desorden general (documentos en suelo, en instalaciones inadecuadas, sin describir, sin mínimo control…).
- Pérdida del principio de procedencia.
- Carencia de práctica habitual de transferencias normalizadas (no respeto al ciclo documental) y/o documentos transferidos sin control ni relaciones descriptivas o identificativas. Percepción del archivo como un problema y no como herramienta de gestión.

1.5 Aportación al proyecto

Al tratarse de un proyecto de migración, en el que el modelo de negocio se ha respetado completamente, las labores de especificación de análisis y especificación de requisitos vinieron dadas en el arranque del proyecto.

No obstante se ha tenido que cuidar que quedase completamente recogido en la nueva aplicación. Además ciertas partes del modelo de datos han tenido que ser adaptadas al modelo de Minerva. Esta era una labor necesaria ya que de otro modo hubiésemos tenido tablas con datos redundantes para los tipos de procedimiento por ejemplo.

Además ha sido necesaria la coordinación con el equipo de desarrollo de Minerva, de modo que la integración entre las dos aplicaciones fuese llevada a cabo con éxito.

1.5.1 El proceso de migración

En este punto se comentan brevemente algunas de las tareas más importantes que han compuesto el proceso de migración. No se entrará en detalles de bajo nivel ya que el objetivo de este trabajo es más, obtener una serie de conclusiones de este proceso, que realizar una especificación exhaustiva del mismo.

- En la aplicación original existía una tabla de códigos para cada desplegable de la aplicación, en nuestro caso hemos incluido la información de estas tablas en una única tabla denominada APPLICATION_RESOURCES que es la gastada en toda la aplicación Minerva. Esto ha obligado a cambiar muchos de ellos para evitar colisiones.

- Optimizada la aplicación para reducir los accesos a base de datos.

- Adaptar la aplicación al nuevo framework, ha requerido de muchas modificaciones en la capa de presentación. Esto es debido a que se han

modificado todas las páginas JSP para utilizar las etiquetas propias de éste. Esto ha ocasionado bastantes problemas con el código Javascript contenido en estas, ya que las nuevas etiquetas no permitían tanta libertad a la hora de definir la id o la clase de cada etiqueta. Esto origina problemas al intentar acceder a las propiedades de los diferentes elementos, que han obligado a revisar y en ocasiones reescribir el código Javascript.

Si bien es cierto, que la utilización de estas nuevas taglibs han reducido considerablemente la cantidad de código javascript incrustado. Esto es posible gracias a que, la modificación de un atributo de estas etiquetas, permite realizar validaciones o añadir funcionalidades, que de otro modo requerirían una función adicional para ello.

- En ambos frameworks se utilizan ficheros XML para establecer las correspondencias entre los Value Objects y los campos de la base de datos. También en estos ficheros se definen las consultas que se utilizan en la aplicación para consulta y modificación.

  No obstante, la sintaxis utilizada para esta labor en ambos frameworks es diferente. Por lo que ha habido que adaptar cuidadosamente estos ficheros.

- Se ha introducido el soporte multilingüe en la aplicación mediante los mecanismos proporcionados por el nuevo framework.

- Las clases de las que heredaban las clases Java originales, han sido cambiadas por versiones más recientes de estas con cambios en sus interfaces, o incluso por otras diferentes, lo que ha obligado a revisarlas y adaptarlas..

- Esta aplicación recibe entregas de expedientes, solicitudes y devoluciones de préstamo a través de Minerva. A su vez envía las respuestas a estas peticiones. Para la integración de ambas aplicaciones ha sido necesaria la coordinación de

nuestro trabajo con el equipo de desarrollo de minerva, ubicado físicamente en Madrid. Para ello la utilización de un CVS ha sido de gran utilidad.

- Se ha planificado y ejecutado el plan de pruebas de la aplicación.

- Se ha efectuado el despliegue en el servidor de pruebas.

- Redacción del manual de usuario.

1.6 Estructura de la tesis

Este documento se compone de 8 capítulos.

Un primer capítulo de introducción en el que se da una visión general del proyecto, se marcan unos objetivos a alcanzar y la justificación de estos objetivos.

El segundo capítulo presenta los requisitos del sistema. En el podemos encontrar una descripción completa del sistema que se ha desarrollado. Mediante la especificación de requisitos funcionales se detallarán las posibles interacciones del usuario con el software.

En el tercer capítulo el lector encontrará algunas de las decisiones de diseño tomadas para la aplicación, y la lógica de la aplicación separada en tres capas, (Presentación, datos, negocio). Al final del capítulo se mostrará la relación entre las diferentes capas.

A continuación tendremos un capítulo en el que enmarcaremos el proyecto dentro de un marco tecnológico. Tanto para la fase de desarrollo como de producción.

En los capítulos quinto y sexto presentamos los problemas encontrados durante el proyecto y se estudia la aplicación de MDA como posible solución a estos.

Para finalizar se enumeran las conclusiones alcanzadas en el capítulo siete.

Por último se incluye la bibliografía y un documento anexo relacionado con el proyecto que podría interesar al lector.

2 REQUISITOS DEL SISTEMA

A lo largo de este capítulo encontramos una descripción resumida de los requisitos del sistema. Esta se centra principalmente en la parte relativa al Servicio Común de Archivo que será el usuario final de esta aplicación, el trabajo realizado por los órganos judiciales desde Cicerone y Nautius no será tratado.

Se trata de una descripción resumida ya que únicamente se detallan los puntos más importantes. Otros como los relativos a la parte de consultas y estadísticas se comentan brevemente, e incluso algunos como la emisión de etiquetas se han omitido. No obstante este capítulo permitirá hacerse una idea clara de la complejidad del sistema. Además servirá para justificar las decisiones de diseño se presentan en el siguiente capítulo.

2.1 Descripción general

Los archivos judiciales son aquellas dependencias en las que se custodian y clasifican los documentos judiciales. De acuerdo con lo establecido por el Real Decreto 937/2003, de 18 de julio, de modernización de los archivos judiciales, existen tres clases de archivo:

- Archivo judicial de gestión.
- Archivo judicial territorial.
- Archivo judicial central.

El archivo judicial de gestión existirá en las oficinas judiciales o unidades análogas y será gestionado por los órganos judiciales o un servicio común que preste servicio a varios juzgados y salas o secciones de tribunales. En este archivo se clasificarán y custodiarán los documentos judiciales generados durante la tramitación de cada expediente.

El archivo judicial territorial se encargará de ordenar la documentación remitida por los responsables de los archivos judiciales de gestión en los casos que establece el Real Decreto. Podrán establecerse varios archivos judiciales territoriales en cada comunidad autónoma.

En el archivo judicial central, adscrito a la sala de Gobierno del Tribunal Supremo, se ordenará la documentación remitida por los responsables de los archivos judiciales de gestión comprendidos en los órganos con jurisdicción en todo el territorio nacional.

El análisis de los requisitos del sistema informático del archivo judicial para la Comunidad Autónoma Valenciana alcanza al archivo de gestión. Se ha tenido en cuenta el funcionamiento de un servicio común de archivo en la Ciudad de la Justicia de Valencia, servicio intermedio que atiende a los órganos judiciales con sede en la ciudad de Valencia.

Al elaborar este documento se ha tenido en cuenta también la normativa vigente en materia de archivos judiciales y, en particular, el Real Decreto 937/2003.

Este análisis del sistema informático del archivo judicial está enfocado, desde un punto de vista orgánico, en dos proyecciones:

- Órgano judicial.
- Servicio de archivo de gestión.

Desde cada una de estas perspectivas se han analizado los procesos que requiere el sistema.

En relación al órgano judicial se contemplan estos procesos, que no detallaremos en este capítulo ya que no es nuestro objetivo realizar una especificación exhaustiva de la aplicación, sino más bien permitir al lector tener una visión global y suficiente.

- Ubicación de expedientes.
- Gestión de entregas que se enviarán al servicio de archivo.
- Incorporación de expedientes devueltos por el servicio de archivo.
- Solicitud de préstamo de expedientes.
- Incorporación de expedientes prestados.
- Devolución de préstamos.

Respecto al servicio de archivo se han previsto los siguientes procesos:

- Ingreso de expedientes enviados por los órganos.
- Registro manual de expedientes y de libros de registro.
- Mantenimiento de datos de los expedientes y de los libros de registro.
- Emisión de etiquetas de identificación de cajas, estanterías y baldas.
- Préstamos a los órganos.
- Recepción de devoluciones de préstamos.
- Envío a otro servicio de archivo.
- Estadísticas.
- Listados.

La relación de procesos que afectan por igual al órgano judicial y al servicio de archivo (y de los cuáles tampoco entraremos en detalles) son:

- Expurgo judicial.
- Movimientos de entrada y salida.

- Mantenimiento de cajas.
- Consultas.
- Mantenimiento de códigos.

En el gráfico siguiente se muestran los procesos de los órganos y servicios que intervienen en el archivo judicial y las relaciones que se establecen entre los mismos:

- Órgano judicial.
- Servicio común de archivo.
- Archivo judicial territorial.
- Junta de expurgo.

En esta representación no se han contemplado las relaciones que los órganos judiciales y el servicio común de archivo mantienen con la junta de expurgo y el archivo judicial territorial, dado que estos organismos quedan al margen del sistema informático del archivo judicial, aunque se han tenido en cuenta las funciones que realizan los archivos judiciales de gestión en relación a aquellos.

La notación seguida es la siguiente:

- Los órganos judiciales que intervienen en el sistema han sido representados como rectángulos nominados en texto azul.
- Cada órgano está rodeado por una serie de óvalos, nominados con texto rojo. Estos representan los diferentes procesos del sistema en que interviene el órgano, en caso de existir.
- Los intercambios de mensajes e información entre órganos y procesos, son representados por flechas nominadas. Estas flechas están orientadas en el sentido que viaja la información.

Por ejemplo podemos apreciar que el proceso de emisión de relación de expurgo, de un órgano judicial genera un listado que es enviado a la Junta de Expurgo.

Otro ejemplo puede ser el proceso de retorno de expediente, que enviará un mensaje que deberá ser procesado por el proceso de registro de devolución de préstamos del Servicio Común de Archivo.

Si observamos la figura detenidamente nos permitirá de un solo vistazo tener una rápida visión completa del sistema, en la que posteriormente iremos profundizando.

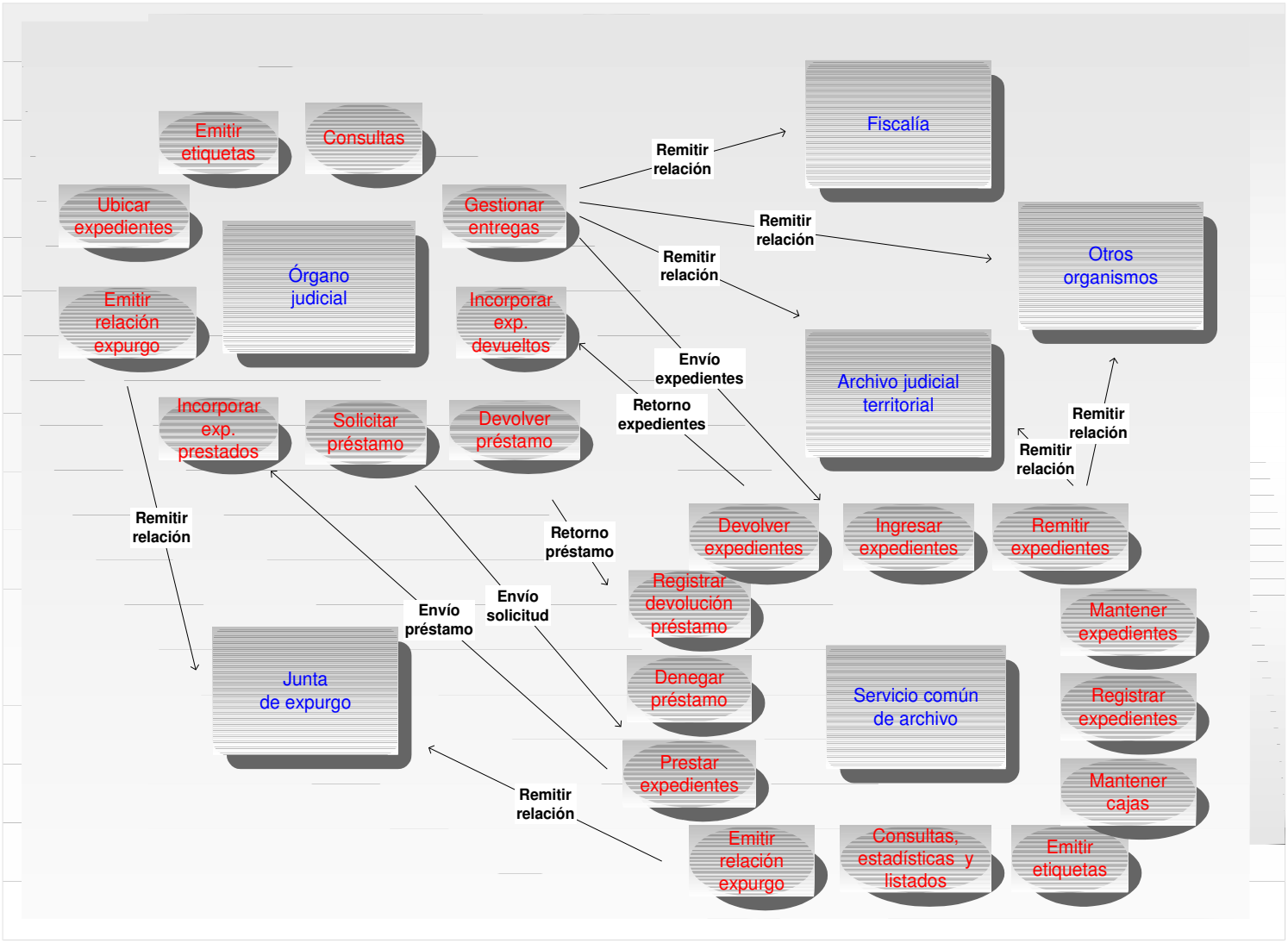

Figura 2. Procesos y servicios Archivo Judicial

2.2 Órgano judicial ubicación de expedientes

En este punto se explica brevemente cual es el proceso de ubicación de expedientes en cada juzgado. Este es un proceso que aunque todavía no hace uso de nuestra aplicación, proporcionará una información fundamental para el correcto uso de esta.

Los órganos judiciales podrán ubicar en el espacio físico que tengan asignado para el archivo judicial de gestión los expedientes que gestionen antes de enviarlos al servicio común de archivo o al archivo judicial territorial.

El proceso de ubicación se gestionará desde cada expediente o por medio de un proceso masivo que permitirá archivar varios expedientes o tomos de éstos.

Al ubicar cada expediente o tomo se podrán indicar estos datos:

- Número de caja.
- Identificación de la estantería.
- Identificación de la balda.

El sistema proporcionará automáticamente el número de caja a través de un contador. Podrá ser modificado por un número anterior al objeto de ubicar el expediente en otra caja.

Si la caja no está ubicada en una estantería se podrán proporcionar los datos de identificación de la estantería y de la balda.

Al registrar la ubicación se podrá indicar algún dato descriptivo del expediente-tomo (p.ej. expediente deteriorado, sin carátulas, etc.).

El proceso masivo requerirá identificar el número de caja en la que se ubicarán los expedientes-tomos. Cuando la caja no estuviera ubicada en una estantería y balda, el sistema permitirá identificarlas.

En relación a la ubicación indicada se introducirán estos datos:

- Año de los expedientes.
- Identificación del órgano: número de exhorto, tipo y número de procedimiento, número de ejecutoria.

Al identificar los expedientes por el número de exhorto, de procedimiento o de ejecutoria se podrán establecer series y menciones concretas (2-5, 8, 10-13).

Los expedientes se podrán desglosar en tomos al realizar el registro de las cajas. El sistema asignará automáticamente un número a cada tomo desglosado y permitirá introducir una descripción identificativa de los mismos.

Al realizar un nuevo registro se capturarán automáticamente algunos datos del anterior registro guardado:

- Año de entrada.
- Tipo de procedimiento.

Las cajas y las estanterías y baldas en las que se ubiquen los expedientes podrán etiquetarse para ser identificadas. Este proceso está descrito en el apartado *Emitir etiquetas de identificación*.

2.3 Servicio común de archivo

2.3.1 Ingresar expedientes

El servicio de archivo recibirá informáticamente las entregas enviadas por los órganos. No se recibirán por esta vía los expedientes antiguos que no estén registrados en el sistema informático de los órganos, los que se encuentren prestados y aquellos que estén ubicados en las dependencias del archivo o pendientes de catalogación en el servicio. El tratamiento de estos expedientes se expone en el capítulo *Registrar expedientes*.

La incorporación de los expedientes al servicio implica que se desarrollen sucesivamente cuatro tareas:

- Cotejo.
- Desglose.
- Ubicación.
- Aceptación.

2.3.1.1 Cotejar expedientes

En primer lugar, el servicio de archivo cotejará las relaciones de entrega con los expedientes enviados por los órganos. Para ello comprobará que la información recibida a través del sistema informático se corresponde con el contenido real de las cajas recibidas.

Entradas

Para ver los expedientes recibidos por el servicio se podrán utilizar estos criterios de acotación:

- Órgano remitente.
- Número / año de la entrega.
- Tipo de entrega (entrada fondos / devolución préstamo).

Será obligatorio indicar el órgano que envía la entrega.

Proceso

El sistema recuperará de la base de datos la información de la entrega. Con ella generará el informe que permitirá al usuario realizar el cotejo de la entrega.

Salida

Se visualizarán los siguientes datos en relación a cada expediente enviado:

- Identificación órgano: número / año de exhorto, último tipo y número / año de procedimiento, tipo y número / año de expediente, número / año de ejecutoria.
- Fecha de entrada.
- Tomo.
- Número de caja.
- Devolución préstamo.

Se mostrará el número de exhorto cuando el expediente de referencia sea un exhorto, el tipo y número de procedimiento cuando se trate de asuntos y recursos en materia civil, penal, social y contencioso-administrativo, el tipo y número de expediente cuando los asuntos no estén identificados por el procedimiento (expedientes gubernativos, de personal, de registro civil, etc.) y el número de ejecutoria cuando el expediente haya sido remitido por un juzgado especializado en la tramitación de ejecutorias.

Los expedientes que componen la entrega se mostrarán ordenados según estos criterios:

- Número de caja en que el órgano ubicó los expedientes.
- Número de exhorto, de procedimiento, de expediente o de ejecutoria dentro de cada caja.

Las relaciones de entrega podrán imprimirse para realizar el cotejo con los expedientes enviados por el órgano. La selección de impresión permitirá imprimir una relación de entrega determinada o todas ellas.

Podrá listarse la ubicación de los expedientes o tomos que fueron prestados para facilitar la localización física de las cajas en las dependencias del archivo.

El listado, que será imprimible, estará ordenado por:

- Identificación de la estantería.
- Identificación de la balda.
- Número de caja.

Se mostrarán estos datos en el listado:

- Identificación órgano: número / año de exhorto, último tipo y número / año de procedimiento, tipo y número / año de expediente, número / año de ejecutoria.
- Tomo.
- Número de caja.
- Identificación de la estantería.
- Identificación de la balda.

2.3.1.1.1 Devolver expedientes

Una vez cotejados, se devolverán los expedientes-tomos que no deban ingresar en el servicio (inexistencia del expediente físico, debe ubicarse en otro servicio de archivo, etc.) al órgano que los envió. Al devolver los expedientes-tomos se indicará el motivo que origina la devolución y la fecha en que se realice quedará automáticamente registrada.

Los expedientes-tomos desaparecerán de la bandeja de entrada al ser devueltos, pero sus datos se podrán consultar ante el eventual requerimiento de información de algún órgano (ver el apartado *Consulta de devoluciones*).

2.3.1.1.2 Listar devoluciones

Se podrá extraer (listar e imprimir) un listado de devolución que se entregará a los órganos que enviaron los expedientes, junto con éstos, en el que constará el motivo y la fecha de devolución de cada expediente.

Entrada

El listado se podrá obtener utilizando los siguientes criterios de acotación:

- Órgano devolución.
- Fecha de devolución (desde / hasta).

Proceso

El sistema leerá la información de la base de datos y generará el informe en relación a los parámetros introducidos.

Salida

El sistema mostrará al usuario un listado con la siguiente información:

- Identificación órgano: número / año de exhorto, último tipo y número / año de procedimiento, tipo y número / año de expediente, número / año de ejecutoria.
- Tomo.
- Número de caja.
- Motivo de devolución.
- Fecha de devolución.

Se emitirán relaciones separadas de devolución por cada uno de los órganos que tramitaron los expedientes. En la cabecera de cada relación se identificará el órgano al que se devuelven los expedientes y el número / año de la entrega.

El listado por órgano se mostrará ordenado por:

- Número de caja

Número de exhorto, de procedimiento, de expediente o de ejecutoria, dentro de cada caja.

2.3.1.2 Desglosar expedientes

Los expedientes podrán ser desglosados en partes o tomos cuando, por razón de volumen, sea necesaria esta división para ubicarlos en cajas.

El desglose se podrá realizar antes de aceptar el expediente. Al desglosar el expediente en tomos, el sistema asignará automáticamente un número a cada tomo desglosado y permitirá introducir una descripción del mismo.

Al desglosar un expediente se mostrarán los tomos en que se encuentra dividido. Los tomos creados podrán ser modificados o eliminados.

Esta operación de desglose también se podrá gestionar accediendo al expediente, una vez haya ingresado en el servicio.

Los tomos de un expediente se podrán ubicar en el servicio de archivo al realizar el desglose, facilitando el número de caja y la identificación de la estantería y de la balda. El proceso para ubicar los tomos es el mismo que el de un expediente y se trata en el siguiente apartado.

2.3.1.3 Ubicar expedientes

Los expedientes que deban ingresar en el servicio podrán ubicarse antes de ser aceptados. El proceso de ubicación permitirá que, simultáneamente, puedan ubicarse varios expedientes o tomos.

Entrada

Se determinará la ubicación de los expedientes-tomos indicando los siguientes datos:

- Número de caja.
- Identificación de la estantería.
- Identificación de la balda.

El sistema proporcionará automáticamente el número de caja a través de un contador cuando el expediente o tomo no se encontrara prestado. Podrá ser modificado por un número anterior al objeto de ubicarlo en otra caja.

Si el expediente o tomo se encontraba en el archivo, el sistema mostrará por defecto la localización física anterior, pero el expediente se podrá ubicar en una caja distinta a la propuesta.

Si la caja no se encuentra ubicada en una estantería se podrán proporcionar los datos de identificación de la estantería y de la balda.

Al registrar la ubicación se podrá indicar algún dato descriptivo del expediente-tomo (p.ej. expediente deteriorado, sin carátulas, etc.).

Proceso

El sistema almacenará toda la información, para que pueda ser consultada a posteriori por cualquier usuario.

Salida

El sistema informará al usuario que el expediente ha sido ubicado correctamente, o de los posibles errores en caso de producirse.

2.3.1.4 Aceptar expedientes

Los expedientes ingresarán en el servicio a través de un proceso de aceptación múltiple, que permitirá la selección y entrada de uno o más asuntos recibidos.

Entrada

En la bandeja de entrada del archivo se encontrarán expedientes-tomos que se reciben por primera vez y otros que fueron prestados y han sido devueltos por los órganos.

Proceso

Los expedientes-tomos prestados no generarán una nueva entrada en el servicio y serán ubicados en la localización que se indique. Cuando no se determine expresamente la localización del expediente-tomo devuelto se mantendrá la ubicación anterior.

Los expedientes desaparecerán de la bandeja de entrada al ser aceptados. Cuando el órgano hubiera desglosado el expediente en tomos, desaparecerán de la bandeja de entrada todos los tomos que componen el expediente.

A cada uno de los asuntos aceptados, salvo que se trate de expedientes prestados (ver el capítulo *registrar devoluciones de préstamos*), se les asignará automáticamente la fecha de entrada en el servicio.

Se generará únicamente una entrada por expediente, aunque el órgano hubiera desglosado en más de un tomo el mismo y cada uno de los tomos será agregado al expediente.

El resto de información que se unirá al expediente aceptado por el servicio es la siguiente:

- Órgano.
- Identificación órgano: número / año de exhorto, último tipo y número / año de procedimiento, tipo y número / año de expediente, número / año de ejecutoria.
- Partes intervinientes.
- Materia del asunto.
- Objeto del asunto.
- Fecha de conclusión de las actuaciones (fecha de estado).

Estos últimos cuatro datos se incluirán en las relaciones documentales que, según exige el Real Decreto 937/2003, deben enviarse al archivo judicial territorial y a la junta de expurgo.

A nivel de expediente o tomo de éste se agregarán los siguientes datos:

- Estado.
- Fecha de estado.
- Número de caja.
- Identificación de la estantería.
- Identificación de la balda.
- Observaciones (ubicación).

Un expediente-tomo podrá encontrarse en uno de estos estados:

- Registrado.
- Ubicado en el servicio.
- Devuelto.
- Prestado.
- Enviado a otro servicio.
- Expurgado.

Posteriormente, se podrá acceder a los expedientes para modificar o actualizar los datos registrados, pero ningún expediente que haya sido aceptado en el servicio podrá ser eliminado. La aceptación por error de un expediente podrá ser enmendada procediendo a su devolución.

Los expedientes que se encuentren archivados en el servicio podrán ser consultados por los órganos o por el propio servicio (ver el capítulo de consultas).

Salida

El sistema emitirá un aviso cuando se trate de aceptar un expediente ya registrado en el servicio de archivo que no se encuentre prestado o devuelto e impedirá su entrada. De esta forma se evitará el registro múltiple de un expediente en la base de datos del archivo. Estos expedientes serán devueltos al órgano que los envió, tal como se expone en el apartado Cotejar expedientes.

2.3.2 Registrar expedientes y libros de registro

Los expedientes que los órganos no envíen informáticamente (asuntos antiguos no registrados en el sistema informático, expedientes prestados y expedientes catalogados o en proceso de catalogación por el servicio) se podrán registrar a través de un proceso ágil en el que se suministrarán los datos imprescindibles para identificar el expediente-tomo y la ubicación física del mismo.

Entrada

En primer lugar se indicará el número de caja en la que se ubicarán los expedientes-tomos. Cuando la caja no estuviera ubicada en una estantería y balda, el sistema permitirá identificarlas.

En relación a cada caja se introducirán estos datos:

- Órgano que envió los expedientes.
- Año de los expedientes.
- Identificación del órgano: número de exhorto, tipo y número de procedimiento, tipo y número de expediente, número de ejecutoria.

La lista de valores del tipo de procedimiento y del tipo de expediente se mostrarán en función de la clase de órgano.

Al identificar los expedientes por el número de exhorto, de procedimiento, de expediente o de ejecutoria se podrán establecer series y menciones concretas (2-5, 8, 10-13).

Los expedientes se podrán desglosar en tomos al realizar el registro de las cajas.

El sistema asignará automáticamente un número a cada tomo desglosado y permitirá introducir una descripción identificativa de los mismos.

En relación a cada expediente-tomo desglosado se registrará:

- Estado (ubicado en el servicio, por defecto).
- Fecha de estado (fecha de sesión, por defecto).

El Real Decreto 937/2003 regula la forma de remisión de las relaciones documentales al archivo judicial territorial y a la junta de expurgo. Para dar cobertura a esta normativa se podrán registrar estos datos adicionales respecto a cada expediente:

- Partes intervinientes.
- Materia del asunto.
- Objeto del asunto.
- Fecha de conclusión de las actuaciones.
- Fecha firmeza.

- Fecha fin.
- Resolución que pone fin al procedimiento.

Como dato adicional de cada expediente también podrá registrarse la fecha de entrada en el servicio de archivo.

Al realizar un nuevo registro se capturarán automáticamente algunos datos del anterior registro guardado:

- Órgano.
- Año de entrada.
- Tipo de procedimiento o Tipo de expediente.

El sistema denegará el registro de aquellos expedientes que se encuentren guardados en la base de datos del servicio.

El proceso de registro de los libros que los órganos envíen al servicio común de archivo es el mismo que está previsto para los expedientes con estas salvedades:

- Los datos que servirán para identificar un libro de registro serán la clase (libro de registro de asuntos penales, libro de sentencias, libro de diligencias previas, etc.), el año y el órgano judicial. En cada registro podrán indicarse adicionalmente observaciones.
- No se realizará ningún tipo de operación de desglose.
- Al registrar un nuevo libro se capturarán automáticamente el órgano y el año indicados en el anterior registro.

Proceso

Se almacenará toda la información en la base de datos.

Salida

El sistema mostrará un mensaje de aviso en caso de que el expediente o libro ya estuviese registrado. Si el registro se lleva a cabo se volverá directamente a la pantalla de registro para continuar con el siguiente registro.

2.3.3 Mantener expedientes y libros de registro

Expedientes

Los datos de los expedientes incorporados al servicio de archivo a través del proceso automático de aceptación y de los expedientes o libros registrados manualmente podrán ser modificados, siempre que el expediente, tomo o libro se encuentren en propiedad del servicio de archivo, es decir cuando no se haya devuelto al órgano que lo envió o esté prestado, enviado a otro servicio o expurgado. En este capítulo se mencionan expresamente los datos que, excepcionalmente, no serán modificables en ningún caso.

Los expedientes y los libros del servicio se podrán eliminar siempre que no concurran las circunstancias expresadas anteriormente. Si el expediente ingresó en el servicio por la vía automática de la aceptación no se podrá eliminar.

Entrada

Se accederá a un expediente suministrando alguno de estos datos:

- Órgano y Número / año de exhorto.
- Tipo y número / año de procedimiento.
- Tipo y número / año de expediente.
- Número / año de ejecutoria.

Será obligatorio indicar el tipo de procedimiento cuando la clase de órgano numere todos los asuntos y recursos según aquel.

Proceso

La información del expediente será modificable en cualquier caso, siempre que el expediente pertenezca al servicio.

A nivel de expediente, de los tomos en que se desglosa el mismo y del libro de registro constarán los siguientes datos:

- Estado.
- Fecha de estado.

El estado y la fecha de estado se actualizarán automáticamente al registrar, ubicar, devolver, prestar, enviar o expurgar un expediente, un tomo o un libro.

Cada movimiento de entrada o salida de un expediente o tomo quedará reflejado (ver el capítulo Movimientos de entrada y salida). Los movimientos generados manualmente podrán modificarse.

No se podrá modificar la fecha de entrada del expediente en el servicio, salvo que hubiera ingresado de forma manual.

Los datos de identificación del expediente (órgano remitente, número de exhorto, tipo y nº de procedimiento, nº de ejecutoria, tipo y nº de expediente), partes intervinientes, descripción del asunto y fecha de conclusión de las actuaciones no se podrán modificar si se hubiera recibido informáticamente.

Los expedientes podrán estar desglosados en tomos y se permitirá la eliminación de éstos, así como la creación de nuevos.

Salida

En cada expediente registrado se mostrará esta información:

- Fecha de entrada.
- Órgano.
- Número de exhorto -para exhortos-
- Último tipo y número de procedimiento –para asuntos y recursos-
- Número de ejecutoria –para ejecutorias de juzgados especializados-.
- Tipo y número de expediente-para el resto de asuntos-.
- Observaciones.

Se podrán visualizar, además, estos datos, que ofrecerán más información del expediente:

- Partes intervinientes.
- Materia del asunto.
- Objeto del asunto.
- Estado.
- Fecha de conclusión de las actuaciones (fecha de estado).
- Fecha firmeza.
- Fecha fin.
- Resolución que pone fin al procedimiento.

Libros

Entrada

El acceso a los datos de un libro de registro se hará identificando el órgano que lo envió y seleccionando el libro que se desea modificar.

Proceso

La información del libro será modificable en cualquier caso, siempre que el expediente pertenezca al servicio.

A nivel de expediente, de los tomos en que se desglosa el mismo y del libro de registro constarán los siguientes datos:

- Estado.
- Fecha de estado.

El estado y la fecha de estado se actualizarán automáticamente al registrar, ubicar, devolver, prestar, enviar o expurgar un expediente, un tomo o un libro.

Salida

En cada libro registrado se mostrará esta información:

- Fecha de entrada.
- Órgano judicial.
- Clase de libro.
- Año.
- Observaciones.

En el expediente, en cada uno de sus tomos y en los libros se reflejarán los datos de ubicación:

- Número de caja.
- Identificación de la estantería.
- Identificación de la balda.
- Observaciones (ubicación).

Esta información será modificable en cualquier caso, siempre que el expediente pertenezca al servicio.

A nivel de expediente, de los tomos en que se desglosa el mismo y del libro de registro constarán los siguientes datos:

- Estado.
- Fecha de estado.

El estado y la fecha de estado se actualizarán automáticamente al registrar, ubicar, devolver, prestar, enviar o expurgar un expediente, un tomo o un libro.

Cada movimiento de entrada o salida de un expediente o tomo quedará reflejado (ver el capítulo *Movimientos de entrada y salida*). Los movimientos generados manualmente podrán modificarse.

2.3.3.1 Devolver expedientes

Los expedientes aceptados por error en el servicio de archivo podrán ser devueltos a los órganos que los enviaron, siempre que no estuvieran prestados, devueltos, enviados a otro servicio o expurgados.

Este proceso se ejecutará en cada expediente y cuando éste o los tomos que lo componen estuvieran en cajas se suprimirá su ubicación.

Al devolver un expediente se indicarán el motivo y la fecha de devolución (fecha de sesión, por defecto), actualizándose automáticamente el estado (devuelto) y la fecha de estado del expediente. En el expediente quedará registrado el movimiento de salida (ver el capítulo *Movimientos de entrada y salida*).

La devolución implicará también la actualización de estos datos en relación a cada uno de los tomos.

Los datos de un asunto devuelto no podrán modificarse.

Se podrá obtener un listado de devolución de expedientes tal como se expone en el apartado *Listar devoluciones*.

La consulta de devoluciones (ver el capítulo de *Consultas*) proporcionará información sobre los expedientes devueltos a los órganos.

2.3.4 Emitir etiquetas de identificación

Existen dos clases de etiquetas que identificarán:

- Cajas y su contenido.
- Estanterías y baldas.

Esta identificación facilitará la búsqueda de los expedientes en la ubicación física determinada por el servicio de archivo.

2.3.5 Prestar expedientes y libros de registro

Los expedientes, tomos o libros archivados en el servicio son prestados a los órganos. Estas operaciones de desarchivo se generan en el servicio a partir de la solicitud de un órgano.

Las solicitudes que reciba el servicio de archivo serán tratadas de dos formas:

- Manualmente, cuando el órgano no haya enviado informáticamente la petición.
- Automáticamente, cuando la solicitud esté enviada con la aplicación del órgano.

El primer supuesto se producirá cuando el servicio de archivo hubiera realizado el préstamo de un expediente o cuando se presten libros de registro. El servicio registrará manualmente el préstamo concedido a través de la funcionalidad específica prevista en el sistema.

En el segundo caso, el órgano que realice la petición registrará la solicitud de préstamo a través de la funcionalidad prevista en el sistema y la enviará informáticamente al servicio de archivo correspondiente. El sistema no permitirá formular solicitudes de préstamo de expedientes o tomos que el órgano no hubiera enviado antes.

Se podrá imprimir una relación de los expedientes o tomos prestados tanto si el préstamo se ha efectuado de forma manual como si se ha hecho de forma automática. El listado se podrá acotar por la fecha (desde / hasta) y hora (desde / hasta) del préstamo y emitirá relaciones separadas por cada órgano al que se realiza el préstamo.

2.3.5.1 Préstamo manual

Entrada préstamo expedientes

Para realizar el préstamo manual de expedientes o tomos se indicará:

- Órgano al que se prestó el expediente.
- Año de los expedientes.
- Identificación del órgano: número de exhorto, tipo y número de procedimiento, tipo y número de expediente, número de ejecutoria.
- Tomo.
- Fecha de préstamo.
- Fecha de solicitud (misma fecha que la de préstamo, por defecto).
- Observaciones.

La lista de valores del tipo de procedimiento se mostrará en función de la clase de órgano.

Al identificar los expedientes por el número de exhorto, de procedimiento o de ejecutoria se podrán establecer series y menciones concretas (2-5, 8, 10-13).

El filtro *tomo* se podrá utilizar en relación a un sólo expediente.

Los valores introducidos en los campos *Órgano*, *Año*, *Tipo de procedimiento* y *Fecha de solicitud* se mantendrán para los expedientes que posteriormente se añadan a la relación.

Entrada préstamo libros

Los datos que se indicarán para incluir libros de registro en la relación de préstamos serán:

- Órgano judicial.
- Clase de libro.
- Año.

Proceso

Cuando el expediente, tomo o libro de registro no estén ubicados en el servicio se impedirá la adhesión de la solicitud a la relación de préstamos hasta que se registren y se indique la ubicación.

Para facilitar la localización de las cajas en el archivo se podrá obtener una relación de los libros de registro solicitados ordenada por la identificación de la estantería, balda y número de caja. Esta relación se podrá imprimir.

Se podrán excluir uno o más expedientes, tomos o libros agregados a la relación y la orden de préstamo sólo afectará a los que se encuentren en aquella.

Salida

Al realizar el préstamo quedará reflejado en cada expediente, tomo o libro de registro el cambio de estado, la fecha de éste y el movimiento de salida (ver el capítulo *Movimientos de entrada y salida*).

2.3.5.2 Préstamo automático

El servicio de archivo recibirá todas las solicitudes y determinará cuáles le corresponde tramitar, por encontrarse los expedientes-tomos ubicados en el servicio, y las que denegará, por tratarse de expedientes prestados, expurgados, devueltos o enviados a otro servicio.

Entrada

Las solicitudes recibidas podrán acotarse por alguno de estos criterios:

- Órgano.
- Número de exhorto –para exhortos-.
- Tipo y nº de procedimiento –para asuntos y recursos-.
- Número de ejecutoria –para ejecutorias-.
- Tipo y nº de expedientes –para el resto de expedientes-.
- Estado (ubicado, por defecto).
- Fecha de solicitud (desde / hasta).
- Urgencia.

En relación a cada solicitud se mostrarán estos datos:

- Órgano.
- Identificación del expediente.
- Tomo.
- Estado.
- Fecha de solicitud.
- Urgencia.

Proceso

Las solicitudes se podrán seleccionar de forma múltiple para proceder a la denegación o admisión.

Salida

La relación de expedientes solicitados se mostrará ordenada por:

- Urgencia.
- Fecha de solicitud.

Para facilitar la localización de las cajas en el archivo se podrá obtener una relación de los expedientes o tomos solicitados ordenada por la identificación de la estantería, balda y número de caja.

Las relaciones de solicitudes, ordenadas por uno u otro concepto, podrán imprimirse.

2.3.5.2.1 Denegar solicitudes

Las solicitudes que no tramite el servicio serán denegadas. Al denegar una solicitud quedará constancia de:

- Motivo de denegación (prestado, devuelto, enviado a otro servicio, expurgado).
- Fecha de denegación (fecha de sesión).
- Observaciones (órgano al que se prestó, devolvió o envió el expediente-tomo y la fecha en que se realizó).

Estos datos se actualizarán automáticamente al denegarse la solicitud. Las observaciones podrán ampliarse con más información que la suministrada por el sistema.

El servicio de archivo tendrá acceso a una consulta en la que podrá ver las solicitudes denegadas.

2.3.5.2.2 Admitir solicitudes

Al dar entrada a una petición se registrarán automáticamente estos datos en el expediente o tomo:

- Estado (prestado).
- Fecha de estado (fecha de sesión).

El préstamo generará un movimiento de salida del expediente o tomo (ver el capítulo *Movimientos de entrada y salida*).

2.3.6 Registrar devoluciones de préstamos

Los expedientes, tomos y libros de registro prestados son devueltos por los órganos al servicio de archivo. Estas operaciones de archivo se generan en el servicio a partir de la devolución de un órgano.

Las devoluciones de préstamos al servicio de archivo serán tratadas de dos formas:

- Manualmente, cuando el órgano no haya devuelto informáticamente el expediente, tomo o libro de registro.
- Automáticamente, cuando la devolución se realice con la aplicación del órgano.

El primer supuesto se producirá cuando el expediente estuviera prestado a un órgano y siempre que se devuelvan libros de registro. El servicio de archivo registrará

manualmente la devolución del préstamo a través de una funcionalidad específicamente destinada a ello.

En el segundo caso, el órgano registrará la devolución de un préstamo a través de los trámites incluidos en la colección normalizada. El órgano enviará informáticamente al servicio de archivo correspondiente las devoluciones generadas y paralelamente remitirá una relación de entrega. El sistema no permitirá la devolución de expedientes o tomos que no hubiera prestado el servicio.

2.3.6.1 Registro manual

Entrada

Para registrar de forma manual la devolución del préstamo de un expediente o tomo se indicará:

- Órgano que devuelve el expediente.
- Año de los expedientes.
- Identificación del órgano: número de exhorto, tipo y número de procedimiento, tipo y número de expediente, número de ejecutoria.
- Tomo.
- Fecha de devolución.
- Observaciones.

La lista de valores del tipo de procedimiento y del tipo de expediente se mostrarán en función de la clase de órgano.

Al identificar los expedientes por el número de exhorto, de procedimiento, de expediente o de ejecutoria se podrán establecer series y menciones concretas (2-5, 8, 10-13).

El filtro *tomo* se podrá utilizar en relación a un solo expediente.

Los valores introducidos en los campos *Órgano*, *Año*, *Tipo de procedimiento*, *Tipo de expediente* y *Fecha de devolución* se mantendrán para los expedientes que posteriormente se añadan a la relación.

Los libros de registro que se incluyan en la relación de devoluciones se identificarán con estos datos:

- Órgano judicial.
- Clase de libro.
- Año.

Cuando el expediente, tomo o libro de registro no esté prestado se impedirá su inclusión en la relación de devoluciones.

Se podrán excluir uno o más expedientes, tomos o libros agregados a la relación y la devolución sólo afectará a los que se encuentren en aquélla.

Para facilitar la localización de las cajas en las que se ubicaron los expedientes, tomos o libros de registro devueltos se podrá obtener un listado ordenado por:

- Identificación de estantería y balda.
- Número de caja.

Proceso

Al registrar la devolución quedará reflejado en cada expediente, tomo o libro el cambio de estado y la fecha de éste y el movimiento de entrada.

Salida

Los expedientes podrán desglosarse y éstos o los tomos se ubicarán en la misma localización física que tenían antes de ser prestados o en una nueva que expresamente se indique. El sistema emitirá un aviso de los tomos desglosados en los que no esté indicada la ubicación física en el archivo.

2.3.6.2 Registro automático

Los expedientes-tomos devueltos formarán parte de las entregas que los órganos realicen al servicio de archivo. La primera tarea del servicio consistirá en cotejar las relaciones de entrega con los expedientes o tomos recibidos. Este proceso se describe en el apartado *Cotejar expedientes*. El desglose, la ubicación y la aceptación de los expedientes devueltos son procesos descritos en este mismo apartado, dónde se aborda la problemática específica de las devoluciones de expedientes prestados.

Al dar entrada a un expediente prestado se registrarán automáticamente los siguientes datos en el expediente o tomo:

- Estado (ubicado).
- Fecha de estado (fecha de sesión).

El movimiento de entrada generado por la devolución del préstamo quedará registrado en el expediente o tomo de referencia (ver el capítulo *Movimientos de entrada y salida*).

2.3.7 Enviar expedientes

Los expedientes y los libros ubicados en el archivo podrán ser enviados a otro servicio o entidad cuando la normativa vigente lo imponga o cuando por razones organizativas, de espacio u otras sea necesario.

Con esta funcionalidad se da cobertura a los envíos que el archivo de gestión debe realizar al territorial y a las entregas que autorice la junta de expurgo, según exige el Real Decreto 937/2003.

El envío de expedientes requiere la ejecución de dos tareas:

- Preparar el envío.
- Enviar a otro servicio / entidad.

2.3.7.1 Preparar el envío

Entrada

Para preparar el envío se utilizará una consulta que dispondrá de estos criterios de acotación:

- Estado (ubicado, por defecto).
- Fecha entrada órgano (desde / hasta).
- Motivo de estado del procedimiento en el órgano (terminado, por defecto).
- Estado del procedimiento en el órgano.
- Fecha de estado (desde / hasta).
- Resolución que pone fin al procedimiento del órgano.

- Fecha firmeza (desde / hasta).
- Fecha fin (desde / hasta).
- Número de caja (desde / hasta).

En la relación de envío podrán incluirse expedientes que estén en situación procesal distinta.

Proceso

Los expedientes seleccionados podrán desglosarse en tomos y tanto los expedientes como los libros podrán ubicarse en cajas. Los procesos para desglosar y ubicar en cajas son los mismos que están previstos para los órganos.

Salida

El resultado de la consulta mostrará estos datos de los expedientes, tomos y libros de registro que cumplan el criterio de acotación establecido:

- Órgano
- Identificación órgano: número / año de exhorto, tipo y número / año de procedimiento, tipo y número / año de expediente, número / año de ejecutoria, clase de libro y año de éste.
- Tomo.
- Estado.
- Órgano préstamo.
- Número de caja.
- Identificación de la estantería.
- Identificación de la balda.

No se mostrarán en el resultado los expedientes, tomos y libros de registro cuyo estado sea devuelto, enviado o expurgado.

Los expedientes y los libros de registro podrán ordenarse por uno de estos criterios:

- Identificación de la estantería y de la balda.
- Número de caja y número de exhorto, de procedimiento, de expediente, de ejecutoria o clase de libro y año de éste dentro de cada caja.

La consulta se podrá imprimir para facilitar la localización de las cajas en las dependencias del archivo y agilizar el envío.

2.3.7.2 Enviar a otro servicio / entidad

Sólo se podrán enviar los expedientes, tomos y libros de registro que se encuentren ubicados en el servicio de archivo, por tanto estarán excluidos de cualquier envío aquellos que estén prestados.

Los expedientes, tomos y libros de registro se seleccionarán de forma individual o múltiple (todos los registros). Cuando se use este método, la selección no afectará a los expedientes o tomos que estén prestados a los órganos.

Al realizar el envío se indicará el servicio o la entidad de destino y si la causa de aquél es el expurgo. Con el envío se actualizarán automáticamente el estado, la fecha de estado y el movimiento de salida de cada expediente, de los tomos si está fragmentado y de los libros de registro. La ubicación en el servicio de los expedientes, tomos y libros de registro (número de caja, identificación de la estantería y de la balda) será eliminada al realizar el envío.

Los expedientes, tomos y libros de registro enviados formarán parte de una entrega identificada por un número que asignará sucesivamente el sistema y el año de envío. Las entregas enviadas podrán ser consultadas posteriormente, pero en ningún caso serán modificables, salvo que se active el proceso de restauración previsto en el sistema. Este proceso permitirá restablecer el estado anterior de uno o más expedientes, tomos y libros de registro.

Enviado un expediente o un libro de registro no se podrá modificar ningún dato y se visualizarán en modo consulta.

Generado el envío se podrá emitir un listado en el que se reflejará el número / año de la entrega y contendrá estos datos por cada expediente, tomo o libro de registro enviado:

- Número de caja.
- Órgano
- Identificación órgano: número / año de exhorto, último tipo y número / año de procedimiento, tipo y número de expediente, número / año de ejecutoria, clase de libro y año de éste.
- Tomo.

La relación de envío que se remitirá al archivo judicial territorial contendrá además esta información en relación a cada expediente:

- Orden jurisdiccional.
- Partes intervinientes.
- Materia del asunto.
- Objeto del asunto.
- Fecha de conclusión de las actuaciones (fecha de estado).

El listado estará ordenado por número de caja y por número de identificación del órgano dentro de cada caja.

2.3.8 Estadísticas

Las estadísticas proporcionarán datos numéricos sobre:

- Expedientes registrados en el servicio.
- Préstamos solicitados por los órganos.

Todas las estadísticas se podrán obtener utilizando las acotaciones previstas en cada una de ellas.

El resultado estadístico se podrá visualizar en pantalla y/o imprimir.

2.3.9 Listados

Los listados proporcionarán datos sobre:

- Ubicación de los expedientes y de los libros en el servicio.
- Expedientes registrados.

Todos los listados se podrán obtener utilizando las acotaciones previstas en cada una de ellos.

La información de cada listado se podrá visualizar en pantalla y / o imprimir.

2.4 Funcionalidades comunes al Órgano Judicial y al Servicio Común de Archivo

2.4.1 Expurgo judicial

En el procedimiento de expurgo judicial regulado en el Real Decreto 937/2003 intervienen:

- Órgano judicial.
- Archivo judicial de gestión.
- Archivo judicial territorial.
- Junta de expurgo.

La intervención del órgano judicial y el servicio de archivo en el procedimiento de expurgo es diferente en función de la ubicación de los documentos judiciales.

Cuando los documentos judiciales radiquen en el archivo judicial de gestión, el órgano judicial o el servicio común de archivo remitirán a la junta de expurgo una relación documental de los expedientes en que se hubiera dictado una resolución que declare la prescripción o la caducidad o en los que haya terminado la ejecución.

Cuando estén ubicados en el archivo judicial territorial, el órgano judicial confirmará el transcurso de los plazos legales de prescripción o caducidad en relación a los expedientes que figuren en la relación remitida por aquél. El servicio común de archivo intervendrá como mero intermediario para remitir la relación enviada por el archivo territorial y comunicar la resolución sobre el transcurso de los plazos legales.

El sistema informático del archivo judicial dispondrá de un proceso de emisión de las relaciones documentales que los órganos o el servicio de archivo enviarán a la junta de expurgo.

Las relaciones estarán identificadas por una descripción y en ellas constará la fecha de envío.

Cada relación estará conformada por los expedientes que se seleccionen a partir de estos criterios:

- Estado procesal del expediente.
- Fecha de estado (desde / hasta).

Podrán incluirse expedientes que se encuentren en diferentes estados.

Por cada expediente se mostrará esta información:

- Órgano
- Identificación órgano: número de exhorto, último tipo y número de procedimiento, tipo y número de expediente, número de ejecutoria.
- Orden jurisdiccional.
- Partes intervinientes.
- Materia del asunto.
- Objeto del asunto.
- Fecha de conclusión de las actuaciones (fecha de estado).

Cuando se autorice el traslado podrá configurarse una entrega, realizar el desglose y ubicar los expedientes o tomos de éstos en cajas numeradas.

La salida del órgano judicial o del servicio de archivo quedará reflejada en cada expediente, suprimiéndose la ubicación que tenían en el archivo. El estado de cada expediente o tomo en el órgano judicial y en el servicio de archivo se actualizará al registrar el movimiento de salida. En caso de error, podrá restaurarse la situación anterior al envío.

2.4.4 Consultas

El servicio de archivo y los órganos judiciales tendrán a su disposición un grupo de consultas para obtener información que se encuentra registrada en el sistema informático.

Las consultas se podrán acotar utilizando una serie de filtros predeterminados en cada una de ellas. Los filtros se podrán utilizar simultáneamente para obtener un resultado más acotado.

Los datos que se mostrarán en el resultado y la ordenación del mismo estarán determinados específicamente en cada tipo de consulta.

Todas las consultas se podrán, además de visualizar en pantalla, imprimir.

Las consultas de la aplicación informática de los órganos judiciales ofrecerán información sobre el destino de los expedientes enviados o retornados por el servicio común de archivo, la fiscalía o cualquier otro organismo.

2.5 Resumen de flujos de información

A continuación se explican a modo de resumen los flujos de información o movimiento de expedientes entre los Órganos Judiciales y el Archivo Judicial de Gestión al que están adscritos, es decir el envío y recepción de expedientes para archivar, y la gestión de préstamos.

2.5.1 Envío y Recepción de Expedientes

Para cada Servicio de Archivo de los que tenga adscritos el Órgano Judicial existirá una entrega activa en la que se incluirán los expedientes que se pretenden enviar. Los expedientes se pueden incluir, bien a través de un trámite desde el propio expediente, o bien desde una funcionalidad específica de la aplicación.

Una vez que se considere completa una entrega se realizará el envío de la misma al Servicio de Archivo, complementando el envío informático con una relación impresa.

El Archivo Judicial, una vez recibida la entrega enviada, deberá proceder al cotejo de las relaciones de entrega con los expedientes enviados. Fruto de este proceso determinará qué expedientes se pueden incorporar al Archivo y cuáles se deben devolver al órgano remitente. Con los expedientes devueltos se remite también una relación impresa para cada órgano origen indicando el motivo de la devolución.

El proceso concluye, en caso de haber devoluciones, con la incorporación de las mismas al Órgano Judicial incluyendo la reubicación si fuese necesaria, o la asignación a otro envío a un Servicio de Archivo diferente.

2.5.2 Gestión de Préstamos

El Órgano Judicial inicia el proceso realizando la solicitud de préstamo al Archivo Judicial, bien a través de un trámite desde el propio expediente, o bien desde una funcionalidad específica de la aplicación. Previamente se puede certificar mediante una consulta si el expediente se encuentra disponible en el Servicio de Archivo y en condiciones de ser prestado.

Una vez preparadas las solicitudes se podrá realizar el envío de las mismas al Archivo Judicial.

El Archivo Judicial, una vez recibidas las solicitudes de préstamo, podrá hacer efectivo el mismo o denegarlo por diversos motivos. En ambos casos se remite al Órgano Judicial la respuesta a la solicitud, y en su caso el expediente.

Una vez recibida la respuesta a la solicitud, el Órgano Judicial realiza la incorporación del préstamo recibido, reubicando si fuese necesario el expediente, y reflejando en la aplicación su nuevo estado.

Cuando en el Órgano Judicial han concluido las actuaciones necesarias, se tiene que devolver el expediente prestado al Servicio de Archivo. Esta devolución se realizará integrando la devolución del préstamo a la entrega activa que exista para enviar al Archivo Judicial, o creando una nueva si no la hubiere.

El último paso del proceso lo constituye la incorporación en el Servicio de Archivo del préstamo devuelto, que se realizará como parte del proceso de aceptación de un envío de expedientes procedente del Órgano Judicial.

2.6 Conclusiones

A lo largo de este capítulo hemos visto un resumen de los requisitos del sistema obtenidos por el analista del proyecto, y que fueron proporcionados al inicio del proyecto. Esto nos permite tener una idea global muy completa del sistema y de lo que esperan los usuarios del mismo.

Es muy importante cuidar esta fase del proceso, ya que de ella depende en gran medida el éxito de nuestra aplicación.

Los objetivos [4] a alcanzar por una buena especificación de requisitos son:

- Ayudar a los clientes a describir claramente lo que se desea obtener mediante un cierto software.
- Ayudar a los desarrolladores a entender qué quiere exactamente el cliente.
- Servir de base para desarrollos de estándares de ERS particulares para cada organización, definiendo su formato y contenido.

Además aportará una serie de ventajas que son:

- Contrato cliente-desarrolladores.
- Reducción del esfuerzo de desarrollo.
- Base para la estimación de costes y planificación.
- Punto de referencia para procesos de verificación y validación.
- Facilitar la transferencia de productos software.
- Base para posibles mejoras.

Hecho esto podemos pasar a ver las decisiones de diseño tomadas para cubrir estos requisitos.

3 DISEÑO DEL SISTEMA.

Una vez presentados los requisitos del sistema, en este capítulo veremos cuáles han sido las principales decisiones de diseño que se han tomado en el desarrollo de la aplicación.

Antes de continuar, es conveniente poner de manifiesto que la notación de diseño utilizada, que veremos a continuación, no respeta la especificación de ningún estándar como pudiera ser UML. Esto es algo que sería necesario corregir para futuros desarrollos, ya que una utilización correcta de UML habría aportado las siguientes ventajas.

UML proporciona una notación estándar y semánticas esenciales para el modelado de un sistema orientado a objetos. Su no utilización, obliga a los revisores a tener que aprender las semánticas y notaciones de la metodología empleada, como paso previo a entender el diseño en sí. Gracias a UML, la comunicación entre diseñadores se facilita, ya que modelando sistemas diferentes puede entender cada uno los diseños de los otros.

Además, la utilización de UML hubiese permitido la utilización de herramientas automáticas, que verificasen las propiedades del modelo, y generasen gran parte del código de la aplicación. De este modo estos diseños dejarían de tener una utilidad meramente documental.

3.1 Decisiones de diseño.

La aplicación de Sistema de Archivos Judiciales es una aplicación web basada en la plataforma J2EE que utiliza el framework Struts 1.2 como espina dorsal de la aplicación. La utilización de Struts 1.2 nos permite separar detalles del código y del control de la aplicación de las vistas de la misma, empleando el Modelo Vista-Controlador en la lógica de presentación.

En este punto resulta apropiado realizar una breve presentación de estos conceptos y proporcionar una breve justificación de su utilización.

En lo referente a la lógica de datos se han empleado los patrones J2EE “Value Object”, “Data Access Object” y “Value Object Assembler”, cuya finalidad es la siguiente,

- Patrón “Value Object”, este patrón se emplea para encapsular los datos de negocio y para enviar/ obtener información hacia/ desde los EJB’s.
- Patrón “Data Access Object”, este patrón se encargará de encapsular todo lo referente a controlar los temas de manejo de conexiones, accesos y almacenamiento de los datos.
- Patrón “Value Object Assembler”, este patrón se emplea para construir los objetos que emplea la aplicación a partir de objetos “Value Object”.

Y en cuanto a la lógica de negocio, se han empleado EJB’s de sesión, a los que accedemos empleando las clases que implementan los patrones de diseño J2EE “Session Façade” y “Service Locator”, que nos ayudan en los siguientes puntos,

- Patrón “Session Façade”, este patrón es empleado para controlar toda la complejidad de las interacciones con los objetos de negocio, proporcionando al cliente un acceso uniforme a los mismos.
- Patrón “Service Locator”, este patrón se encargará de abstraer el uso de JNDI y la complejidad de la creación, conexión, etc de EJB’s.

3.1.1 ¿Por qué J2EE?

En las siguientes líneas queda justificada la utilización de J3EE. [3]

Con la aparición de las aplicaciones web que dependían cada vez más de tecnologías de servidor como es el middleware, los departamentos de informática de las empresas necesitaban alguna forma sustentable de desarrollar aplicaciones y el middleware relacionado que fueran portables y escalables.

Era necesario diseñar estas aplicaciones de forma que pudieran atender a miles de usuarios en forma simultánea, 24 horas, sin ningún tipo de inactividad. J2EE simplifica la creación de las aplicaciones empresariales, ya que la funcionalidad se encapsula en los componentes de J2EE. Esto permite a los diseñadores y programadores dividir la aplicación de acuerdo con sus funciones.

J2EE se ha convertido en el entorno empresarial estándar lo que permite a las empresas estar seguras que los productos servidores que compran están soportados dentro de J2EE. Esto significa también que no están atadas a un solo proveedor lo que permite elegir en base a su eficiencia y efectividad.

J2EE es una tecnología versátil, ya que los componentes de las aplicaciones que se construyen con J2EE se comunican entre sí mediante métodos estándar como son HTTP, SSL, CML, RMI e IIOP.

La utilización de Java asegura que se poseerán todas las interfaces y bibliotecas necesarias para gestionar cuestiones complejas.

3.1.2 Presentación J2EE

J2EE [2] es una plataforma de programación—parte de la Plataforma Java—para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. La plataforma Java EE está definida por una *especificación*. Similar a otras especificaciones del Java Community Process, Java EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son *conformes a Java EE*; no obstante sin un estándar de ISO o ECMA.

Java EE incluye varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para Java EE para componentes. Estas incluyen Enterprise JavaBeans, servlets, portlets (siguiendo la especificación de Portlets Java), JavaServer Pages y varias tecnologías de servicios web. Esto permite al desarrollador crear una Aplicación de Empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel.

Uno de los beneficios de Java EE como plataforma es que es posible empezar con poco o ningún coste. La implementación Java EE de Sun Microsystems puede ser descargada gratuitamente, y hay muchas herramientas de código abierto disponibles para extender la plataforma o para simplificar el desarrollo.

Ahora veremos unos ejemplos de herramientas de desarrollo Java de código abierto que han sido utilizadas en el desarrollo. Comenzando por Struts, un framework para desarrollar aplicaciones web EE conforme al modelo MVC.

3.1.3Modelo Vista Controlador

Un diagrama sencillo que muestra la relación entre el modelo, la vista y el controlador.

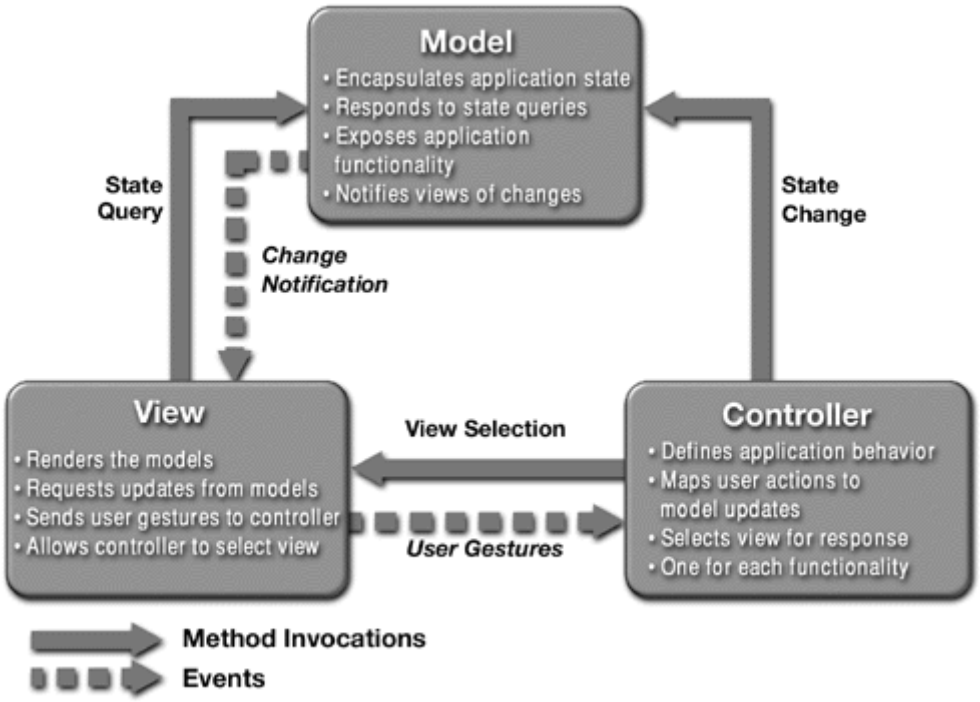

Figura 3. Relaciones entre las capas

MVC [2] es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio y el controlador es el responsable de recibir los eventos de entrada desde la vista.

Descripción del patrón

- **Modelo**: Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o importes en un carrito de la compra.
- **Vista**: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- **Controlador**: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos. En MVC corresponde al modelo.

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. *Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.*

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

3.1.4 Struts

Struts [2] es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma J2EE (Java 2, Enterprise Edition). Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Foundation, pero actualmente es un proyecto independiente conocido como Apache Struts.

Struts permite reducir el tiempo de desarrollo. Su carácter de "*software libre*" y su compatibilidad con todas las plataformas en que Java Entreprise esté disponible, lo convierte en una herramienta altamente disponible.

Entre las características de Struts se pueden mencionar:

- Configuración del control centralizada.

- Interrelaciones entre acciones y página u otras acciones se especifican por tablas XML en lugar de codificarlas en los programas o páginas.
- Componentes de aplicación, que son el mecanismo para compartir información bidireccionalmente entre el usuario de la aplicación y las acciones del modelo.
- Librerías de entidades para facilitar la mayoría de las operaciones que generalmente realizan las páginas JSP.
- Struts contiene herramientas para validación de campos de plantillas bajo varios esquemas que van desde validaciones locales en la página (en javaScript) hasta las validaciones de fondo hechas a nivel de las acciones.

Struts permite que el desarrollador se concentre en el diseño de aplicaciones complejas como una serie simple de componentes del Modelo y de la vista intercomunicados por un control centralizado. Diseñando de esta manera puede obtenerse una aplicación más consistente y más fácil de mantener.

3.1.5 Apache Ant

Apache Ant [2] es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build). Es similar a Make pero sin las engorrosas dependencias del sistema operativo.

Esta herramienta, hecha en Java, tiene la ventaja de no depender de las órdenes de shell de cada sistema operativo, sino que se basa en archivos de configuración XML y clases Java para la realización de las distintas tareas, siendo idónea como solución multi-plataforma.

Ant es un proyecto de código abierto de la Apache Software Foundation.

A continuación se muestra un archivo de ejemplo (build.xml) para una aplicación "Hola mundo" en Java. El archivo define tres objetivos - clean, compile y jar, cada uno de los cuales tiene una descripción asociada. El objetivo jar lista el objetivo compile como dependencia. Esto le dice a ANT que, antes de empezar el objetivo jar, debe completar el objetivo compile.

Dentro de cada objetivo están las acciones que debe tomar ANT para construir el objetivo. Por ejemplo, para construir el objetivo compile ANT debe primero crear un directorio llamado classes (ANT sólo lo hará si éste no existe previamente) y luego llamar al compilador de Java.

```
<?xml version="1.0"?>
<project name="Hello" default="compile">
    <target name="clean" description="borrar archivos temporales">
        <delete dir="classes"/>
    </target>
    <target name="compile"
     description="compilar el código java a un archivo class">
        <mkdir dir="classes"/>
        <javac srcdir="." destdir="classes"/>
    </target>
    <target name="jar" depends="compile"
     description="crear un archivo Jar para la aplicación">
        <jar destfile="hello.jar">
            <fileset dir="classes" includes="**/*.class"/>
            <manifest>
                <attribute name="Main-Class" value="HelloProgram"/>
            </manifest>
        </jar>
    </target>
</project>
```

3.2 Lógica de presentación

En el siguiente apartado se mostrarán todas las clases que se han desarrollado para realizar el diseño del modelo vista-controlador utilizando el framework *Struts 1.2* para la aplicación del sistema de archivos.

El framework de *Struts 1.2* utiliza clases que extienden de las clases "*ActionForm*" para almacenar los datos que un usuario envía en un formulario y proporcionárselos a las clases que extienden de las clases "*Action*" y "*DispatchAction*", que serán los encargados de utilizar estos datos para llamar a los objetos de negocio realizar una serie de operaciones y devolver un resultado determinado. Para ello *Struts 1.2* dispone de un servlet que a partir de la configuración indicada en el fichero "struts-config.xml", que asocia las clases "*ActionForm*" con las clases "*Action*" o "*DispatchAction*", se encargará de recibir las solicitudes del usuario (envío de formularios, acceso a enlaces, etc), construir los objetos "*ActionForm*" e invocar a los métodos apropiados de las objetos "*Action*" o "*DispatchAction*" para que accedan a los objetos de negocio, realicen una serie de operaciones y proporcionen los resultados correspondientes que se mostrarán en una vista determinada.

El funcionamiento del controlador de *Struts 1.2* (*servlet*) junto con las clases "*ActionForm*", "*Action*" y "*DispatchAction*" se pueden apreciar en la siguiente figura.

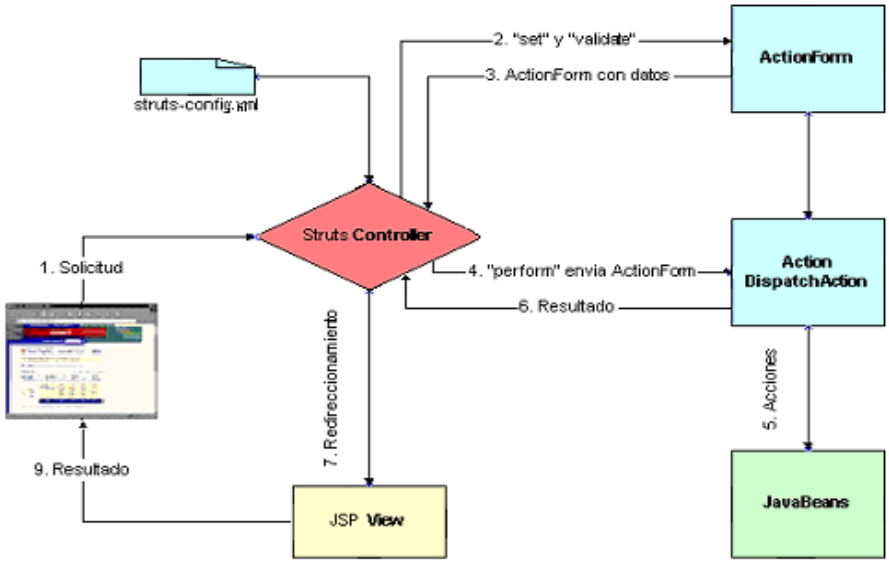

Figura 4. Utilización ActionForms, Action y DispatchAction

3.2.1 Clases Action Form

En esta aplicación necesitamos tanto mostrar como recoger información de los usuarios por lo que para ello utilizamos una serie de páginas JSP formada por una serie de etiquetas personalizadas (Taglibs). La utilización de estas etiquetas aporta las siguientes ventajas:

- facilitar el aprendizaje.
- facilitar el mantenimiento.
- fomentar la modularidad y la reutilización.
- simplificar el código y reducir el número de líneas necesarias.

Para dar la apariencia que se marca en la guía de estilo de aplicaciones del ministerio de justicia se han utilizado las horas de estilo proporcionadas por este, con alguna aportación particular para este proyecto.

En la siguiente figura podemos ver el resultado que obtiene el usuario en su navegador.

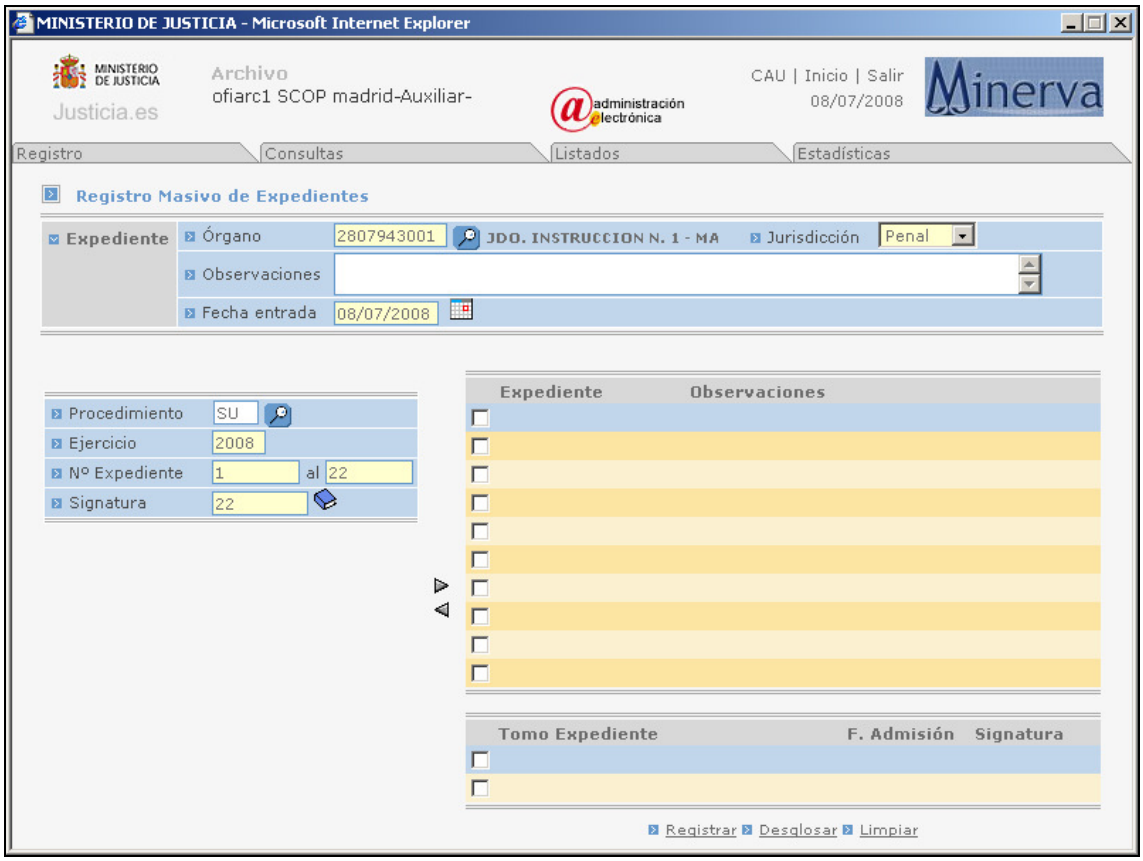

Figura 4. Ejemplo de formulario de recogida de datos

Como comentamos en el punto anterior para recoger esta información utilizaremos una serie de clases Java que pasamos a ver a continuación.

En el siguiente diagrama de clases se pueden apreciar las clases que heredan de "*ActionForm*" y que han sido diseñadas para el realizar la aplicación de Sistema de Archivo.

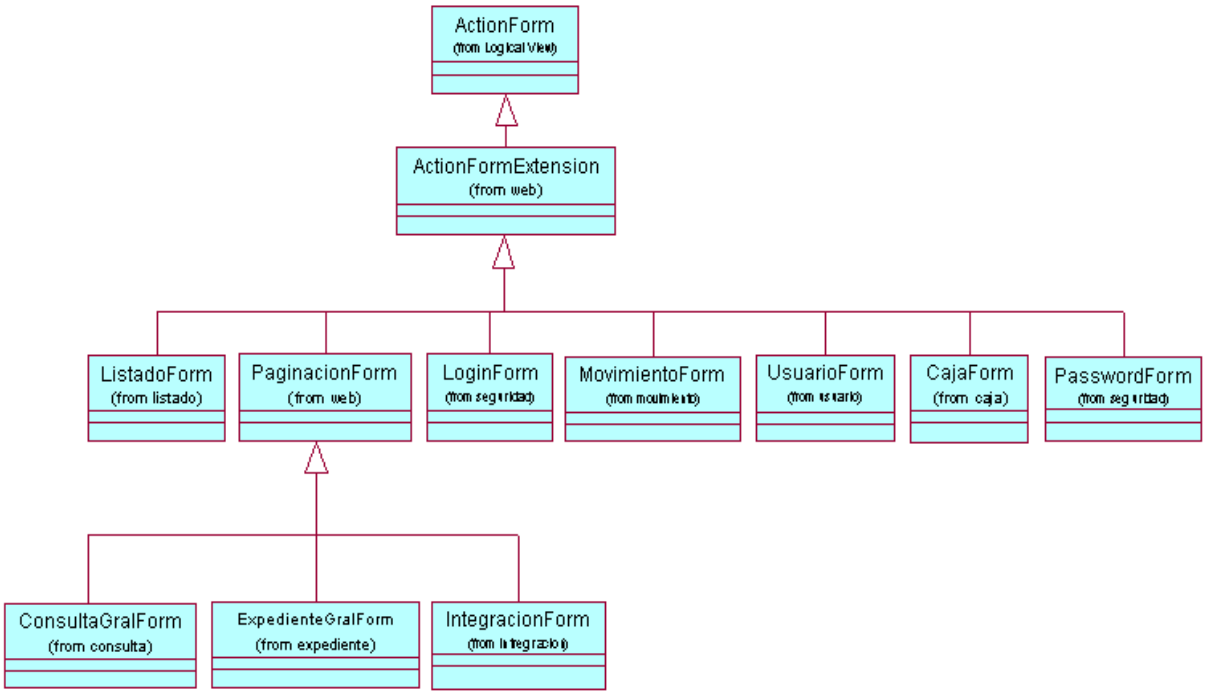

Figura 5. Diagrama de clases ActionForm

Como se puede apreciar en el diagrama de clases anterior, la clase "ActionFormExtension" hereda de la clase "ActionForm", dándole más funcionalidad a esta clase base. En este caso en concreto, la clase abstracta "*ActionFormExtension*", que se encuentra en el framework de SEINTEX, en el JAR **SeintexBaseWEB**, proporciona un "*log*" para mantener un registro de las operaciones que se están realizando sobre las clases "*ActionForm*" que hereden de esta.

La finalidad de las clases que heredan de la clase abstracta "ActionFormExtension" es obtener y establecer los datos que se mostrarán y se introducirán en la capa de vista de la aplicación, para ello deberán implementar los métodos "*getter*" y "*setter*" correspondientes que se encargarán de obtener y establecer cada una de las entradas de los diferentes formularios (JSP) que tengan asociados de la capa de vista. Además también es posible implementar los métodos "*validate*" y "*reset*" para validar e inicializar respectivamente los campos de un formulario. De esta forma los datos que hay en un formulario se pueden obtener y establecer a partir de la clase "*ActionForm*" correspondiente que se haya desarrollado para cada *JSP*.

Las clases que extienden la clase "*ActionFormExtension*", y por tanto la clase "*ActionForm*", se pueden agrupar en la siguiente clasificación,

- **Expediente**, clase "*ActionForm*" utilizada para obtener los datos del formulario que se encarga de manipular expedientes, libros, tomos y proporcionar los mismos a las clases que extienden "*Action*" y "*DispatchAction*". La clase que se emplea para manejar toda esta información es "*ExpedienteGralForm*".

- **Caja**, clase "*ActionForm*" utilizada para obtener los datos que se escriben en el formulario encargado de manipular las cajas en las que se encuentran los expedientes, crear nuevas cajas, eliminar cajas, reubicar cajas, etc. Esta clase también manejará los datos empleados para generar las etiquetas de las cajas, estanterías y baldas. La clase que se emplea para manejar toda esta información es "*CajaForm*".

- **Movimiento**, clase "*ActionForm*" que se encarga de obtener los datos que se escriben en el formulario de control de movimientos que se realizan sobre un expediente. La clase que se emplea para manejar estos datos es "*MovimientoForm*".

- **Listado**, clase "*ActionForm*" que se encarga de recibir los datos del formulario que se utiliza para determinar los valores que se utilizarán en los criterios de acotación empleados para realizar listados de los datos que se utilizan en la aplicación del Sistema de Archivo. La clase que se utiliza para manejar toda esta información es "*ListadoForm*".

- **Consulta**, clase "*ActionForm*" que se encarga de obtener los datos de los formularios que se utilizan para determinar los valores que se emplearán en los criterios de acotación utilizados para realizar las consultas sobre los diferentes datos que nos interesen de la aplicación del Sistema de Archivo. La clase que se utiliza para manejar toda esta información es "*ConsultaGralForm*".

- **Integración**, clase "*ActionForm*" que se encarga de obtener los datos de los formularios que se utilizan para realizar las operaciones de integración de la aplicación de Sistema de Archivo, tales como enviar expedientes, recibir solicitudes, recibir fondos, etc. La clase que se utiliza para manejar toda esta información es "*IntegracionForm*".

- **Usuario**, clase "*ActionForm*" que se encarga de obtener los datos de los formularios que se utilizan para realizar un mantenimiento de usuarios, es decir, crear nuevos usuarios, borrar usuarios, dar permisos a un usuario, etc. La clase que se emplea para obtener y acceder a esta información que se encuentra en los formularios es "*UsuarioForm*".

- **Acceso**, clases "*ActionForm*" que se encarga de obtener los datos del formulario empleado por los usuarios para acceder a la aplicación del Sistema de Archivo (nombre y clave) y manejar todo lo referente a cambio de clave, notificación de errores durante el acceso a la aplicación debido a usuario no valido o clave no valida, etc. Las clases que se emplean para obtener y acceder a esta información son "*LoginForm*" y "*PaswordForm*" que están incluidas en el *JAR* **SeintexBaseWEB** en el framework de SEINTEX.

3.2.2 Clases Action

En el siguiente diagrama de clases se pueden apreciar las clases que heredan de "*Action*" y que han sido diseñadas para el realizar la aplicación de Sistema de Archivo.

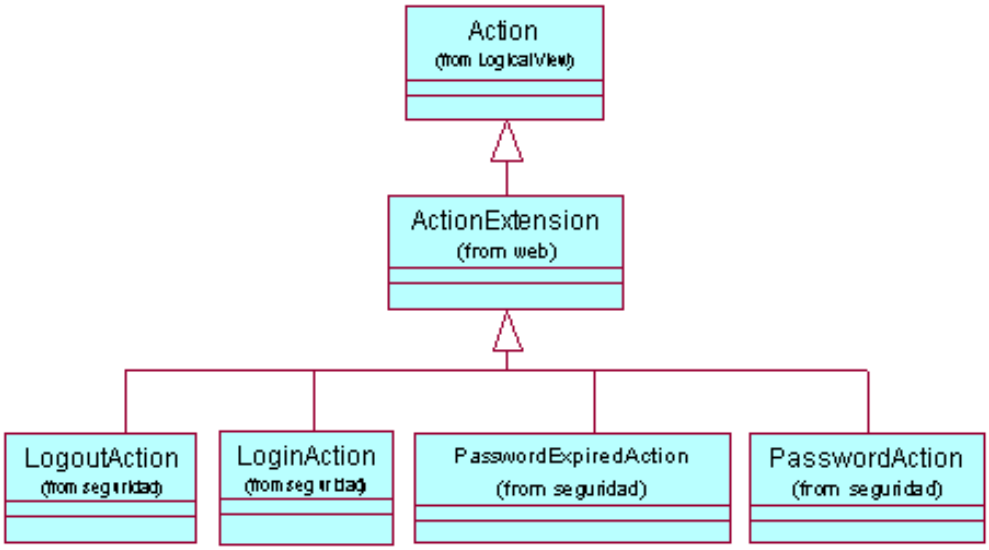

Figura 6. Diagrama de clases Action

Como se puede apreciar en el diagrama de clases, la clase abstracta "*ActionExtension*" hereda de la clase "*Action*" dando más funcionalidad a esta clase, proporciona un "log" para mantener un registro de las operaciones que se están realizando sobre las clases "*Action*", proporciona métodos para acceder al entorno de aplicación y de sesión, métodos para realizar un registro de los accesos a esta clase, etc. Esta clase,.al igual que la clase "*ActionForm*", está incluida en el *JAR* **SeintexBaseWEB** en el framework de SEINTEX.

El resto de clases que se muestran en el diagrama, "*LoginAction*", "*LogoutAction*", "*PasswordExpiredAction*" y "*PasswordAction*", extienden la clase "*ActionExtension*", con lo cual deberán implementar el método "*executeAction*", que será el método invocado por el controlador de *Struts 1.2* cada vez que el usuario genere un evento que requiera de la participación de una de estas tres clases (esta relación esta codificada en el fichero "*struts-config.xml*").

La finalidad de estas clases es la siguiente,

- **LoginAction**, se encarga de recibir los datos de acceso del usuario a través de un objeto de la clase "*LoginForm*" y de llamar a los objetos que ejecuten la lógica de negocio que se encargará de comprobar que el usuario puede acceder a la

aplicación, ver si el usuario tiene los permisos requeridos, guardar un registro del acceso y controlar las sesiones del usuario.

- **LogoutAction**, se encarga de cerrar la sesión y de registrar que el usuario a abandonado la aplicación.

- **PasswordAction**, se encargará de recibir los datos de la nueva clave para un usuario a través de un objeto de la clase "*PasswordForm*" y de llamar a los objetos que ejecutarán la lógica de negocio que realizará el cambio del password para un usuario.

- **PasswordExpiredAction**, se encargará de indicar al usuario que su clave ha caducado y de llamar a los objetos que ejecuten la lógica de negocio encargada de permitir al usuario que cambie su clave y proporcionar acceso al usuario a la aplicación.

3.2.3 Clases DispatchAction.

En el siguiente diagrama de clases se pueden apreciar las clases que heredan de "*DispatchAction*" y que han sido diseñadas para el realizar la aplicación de Sistema de Archivo.

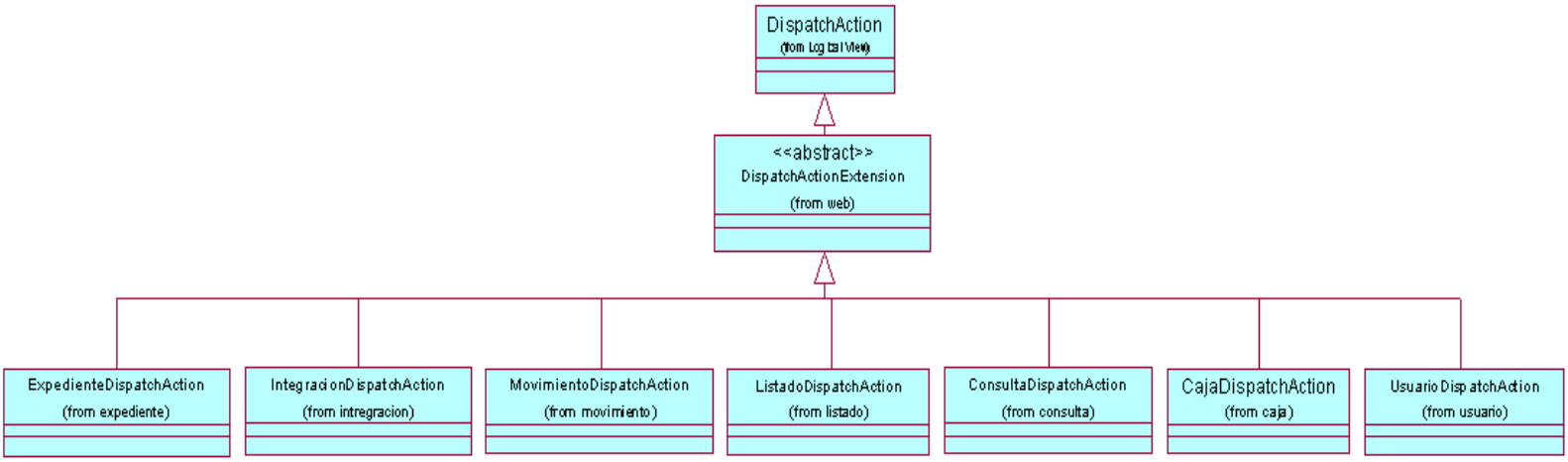

Figura 7. Diagrama de clases DispatchAction

Como se puede apreciar en el diagrama de clases, la clase abstracta "*DispatchActionExtension*" hereda de la clase "*DispatchAction*" al igual que en los otro casos para aportar más funcionalidad a esta. Esta aporta los métodos necesarios para implementar un mecanismo de control de errores, proporciona métodos para acceder al entorno de aplicación y de sesión, métodos para realizar un registro de los accesos a esta clase, etc. Esta clase también está incluida en el *JAR* **SeintexBaseWEB** del framework de SEINTEX.

El resto de clases que se muestran en el diagrama, "*ExpedienteDispatchAction*", "*ListadoDispatchAction*", "*CajaDispatchAction* "*IntegracionDispatchAction*", "*MovimientoDispatchAction*" y "*ConsultaDispatchAction*" extienden la clase "*DispatchActionExtension*". La principal diferencia de estas clases con las clases que extienden a "ActionExtension" reside en que las primeras en lugar de tener un único método "*executeAction*", agrupan un conjunto de métodos que mantienen una relación y que el controlador de Struts deberá ser capaz de invocar.

La descripción de los métodos que agrupan cada una de estas clases son las siguientes,

- **ExpedienteDispatchAction**, esta clase contiene todos aquellos métodos que tienen que ver con el uso de la lógica de negocios que se encarga de editar expedientes, asuntos, libros, tomos, operaciones de inserción/ eliminación de expedientes, desglose de expedientes en tomos, etc.

- **ListadoDispatchAction**, esta clase contiene todos aquellos métodos que tienen que ver con el uso de la lógica de negocios que se encarga de generar listados de todo tipo de datos que maneja la aplicación de Sistema de Archivo como listados de expedientes, de solicitudes de prestamos, de envíos de expedientes, etc.

- **CajaDispatchAction**, esta clase contiene todos aquellos métodos que tienen que ver con el uso de la lógica de negocios que se encarga de crear nuevas cajas, ubicar cajas, etc.

- **ConsultaDispatchAction**, esta clase contiene todos aquellos métodos que tienen que ver con el uso de la lógica de negocios que se encarga de realizar consultas a la base de datos, como obtener todos los expedientes que han sido prestados, los expedientes almacenados en una determinada caja, los expedientes que han entrado al sistema en una determinada fecha, etc.

- **MovimientoDispatchAction**, esta clase contiene todos aquellos métodos que tienen que ver con el uso de la lógica de negocios que se encarga de las operaciones de movimientos de expedientes, de forma que con sus métodos se podrá mantener un registro del movimiento de los expedientes. Estas operaciones pueden ser generar un movimiento de préstamo para un determinado expediente, tomo o libro, generar un movimiento de devolución de un préstamo, etc.

- **IntegracionDispatchAction**, esta clase contiene todos aquellos métodos que tienen que ver con el uso de la lógica de negocios que se encarga de controlar los buzones de entrada y salida, en los que se reciben solicitudes de préstamo, inserción de nuevos expedientes, devolución de prestamos, etc y envían los préstamos solicitados, los rechazos de solicitudes, las devoluciones de expedientes, etc.

- **UsuarioDispatchAction**, esta clase contiene todos aquellos métodos que tienen que ver con el uso de la lógica de negocios que se encarga de controlar las operaciones de administración de los usuarios, tales como dar de alta un nuevo usuario, dar de baja un usuarios, cambiar permisos de un usuarios.

3.3 ACCESO A DATOS

En este apartado se describen todas las clases que se han desarrollado para controlar el acceso a los datos y los objetos que se han utilizado para intercambiar información entre la lógica de negocio y los usuarios y viceversa.

3.3.1 Value Objects

Los objetos que se han creado siguiendo el patrón "*Value Object*" son empleados para intercambiar información entre los objetos de la lógica de negocio (*EJB's*) y los usuarios o viceversa. Las clases que codificarán estos objetos constan de métodos "*get*" y "*set*" para cada uno de los atributos que se almacenarán en el "*Value Object*" del cual queremos obtener la información que nos interese.

Normalmente, cuando un usuario necesita acceder a una cierta información que le proporciona un *EJB* de la lógica de negocio, accederá a más de un valor que le proporcionará este objeto, con lo cual estos accesos afectarán al rendimiento de la red, degradándola cuantos más accesos se realicen por parte del cliente al *EJB*.

Para solucionar este problema, de degradación del rendimiento de la red, se utilizan las clases que implementan el patrón "*Value Object*", que permiten encapsular en un único objeto varios atributos que proporciona un determinado *EJB*, de forma que con un solo acceso a un método del *EJB* este le devolverá al cliente un conjunto de atributos agrupados en un "*Value Object*" y viceversa, el cliente podrá proporcionarle varios atributos al *EJB* de la lógica de negocio con un solo "*Value Object*", reduciendo en ambos casos el tráfico en la red.

3.3.1.1 Clases generales

Las clases que se describen en este punto, han sido utilizadas para encapsular aquellos atributos y métodos comunes a algunas de las clases que implementan el patrón "*Value Object*", estas clases genéricas son,

- **DataVO**, clase genérica serializable que implementa el patrón "*Value Object*" y que aporta el atributo identificador de sesión. De esta clase extenderán todas aquellas clases que se empleen para acceder y proporcionar información que proporciona y recibe los componentes de la lógica de negocio.
- **ExpedienteVO**, clase básica que extiende a la clase *DataVO* y que contiene los métodos y atributos comunes a las clases *AsuntoVO* y *LibroVO*. Además esta clase implementa el interfaz *ExpedienteGral*.
- **BuzonVO**, clase básica que extiende a la clase *DataVO* y que contiene los métodos y atributos comunes a las clases *BEntradaVO* y *BSalidaVO*.
- **Usuario**, clase básica que extiende a la clase *DataVO*, implementa el interfaz Usuario y que contiene los métodos y atributos necesarios para trabajar con los datos de cualquier usuario.

La clases *DataVO* se encuentra en el framework de SEINTEX, encapsulada en el *JAR* SeintexBaseSEG.

3.3.1.2 Interfaces

Los siguientes interfaces se utilizan para que todas las clases que trabajen con un mismo tipo de datos empleen los mismos métodos, con los mismos tipos de parámetros de entrada y valores devueltos, tal que cualquier cambio en el tipo de uno de estos parámetros o valores devueltos nos obligue a actualizar todas las clases que implementen estos interfaces.

Los interfaces que se emplean para conseguir esta finalidad son los siguientes,

- **ExpedienteGral**, interfaz que describe los métodos comunes que deberán implementar todas aquellas clases que trabajen con los datos de un expediente (*ExpedienteVO* y *ExpedienteObj*).

- **Asunto**, interfaz que describe los métodos comunes que deberán implementar todas aquellas clases que trabajen con los datos de un asunto (*AsuntoVO* y *AsuntoObj*).

- **Libro**, interfaz que describe los métodos comunes que deberán implementar todas aquellas clases que trabajen con los datos de un libro (*LibroVO* y *LibroObj*).

- **Tomo**, interfaz que describe los métodos comunes que deberán implementar todas aquellas clases que trabajen con los datos de un tomo (*TomoVO* y *TomoObj*).

- **Caja**, interfaz que describe los métodos comunes que deberán implementar todas aquellas clases que trabajen con los datos de un caja (*CajaVO* y *CajaObj*).

- **Interviniente**, interfaz que describe los métodos comunes que deberán implementar todas aquellas clases que trabajen con los datos de un interviniente (*IntervinienteVO* y *IntervinienteObj*).

- **Movimiento**, interfaz que describe los métodos comunes que deberán implementar todas aquellas clases que trabajen con los datos de un movimiento (*MovimientoVO* y *MovimientoObj*).

- **Usuario**, interfaz que describe los métodos comunes que deberán implementa aquellas clases que trabajen con los datos de un usuario (Clase general Usuario).

3.3.1.3 Clases Value Object

Las clases que se han creado siguiendo el patrón "Value Object", que extienden las clases generales y que implementa los interfaces que se han especificado en los puntos anteriores, se describen a continuación.

- **AsuntoVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que caracterizan a un determinado asunto (expediente) del Sistema de Archivo y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encarga del acceso a la base de datos. Esta clase extiende a la clase *ExpedienteVO* e implementa el interfaz *Asunto*.

  Hay que tener en cuenta que un asunto puede constar de varios tomos o ser tomo único, puede tener uno o varios intervinientes, pertenecerá a un órgano determinado.
- **LibroVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía todos los atributos relacionados con un determinado libro (expediente) del Sistema de Archivo y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que controlará el acceso a la base de datos para acceder a dicha información. Esta clase extiende a la clase *ExpedienteVO* e implementa el interfaz *Libro*.

  Un libro solo constará de un tomo, pertenecerá a un órgano no tendrá intervinientes.

- **TomoVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía todos los atributos relacionados con un determinado tomo, en los que se divide un expediente (asunto o libro), del Sistema de Archivo, y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende a la clase *DataVO* e implementa los interfaces *Asunto* y *ExpedienteGral*.

  Todos los expedientes que hay en la aplicación de Sistema de Archivo, se encuentran divididos en tomos. Un asunto, como se ha explicado anteriormente, se divide en uno o varios tomos y un libro consta de un único tomo, cada tomo será almacenado en una caja del Sistema de Archivo, además, sobre los diferentes tomos se pueden haber realizado uno o varios movimientos.

- **CajaVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que caracterizan a un determinada caja del Sistema de Archivo y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende la clase *DataVO* e implementa el interfaz *Caja*.

  Una caja contendrá uno o varios expedientes divididos en sus correspondientes tomos, por lo tanto una caja almacenará uno o varios tomos. Aunque normalmente una caja almacenará expedientes completos (asuntos y libros), es decir en una misma caja normalmente tendremos todos los tomos en los que se divide un expediente, cabe la posibilidad de que tomos pertenecientes a un mismo expediente se encuentren en cajas diferentes.

- **IntervinienteVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que caracterizan a un determinada persona que participa en un expediente del Sistema de Archivo y que proporcionará/ recibirá un EJB de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende a la clase *Usuario* e implementa el interfaz *Interviniente*.

  Los intervinientes solo tendrán sentido en los asuntos, de forma que un asunto podrá tener uno o varios intervinientes y un mismo interviniente puede participar en varios asuntos.

- **OrganoVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que caracterizan a un determinada órgano judicial y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende a la clase *DataVO*.

  Cualquier tomo en los que se divide un expediente, ya sea asunto o libro, pertenece a un órgano, cualquier tomo que se saque del Sistema de Archivo

tendrá un órgano de destino y cualquier órgano que se encuentre en el sistema de archivo tendrá un órgano identificador del archivo en el que se encuentre.

- **MovimientoVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que caracterizan a un determinado movimiento de un expediente y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende a la clase *DataVO* e implementa el interfaz *Movimiento*.

  Un movimiento consiste en cualquier operación que implique un cambio en la ubicación de un tomo (ya sea un tomo correspondiente a un expediente o un libro), las operaciones de movimiento que se podrán realizar son las de préstamo de tomos, registro de un expediente, devolución de un tomo prestado, reubicación de un tomo en el que se divide un expediente (asunto o libro), etc.

- **BEntradaVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que caracterizan a la recepción de alguna petición de un órgano judicial para realizar operaciones como registro de expedientes, solicitudes de préstamo de expedientes, devolución de préstamo de expedientes, etc., y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende la clase BuzonVO.

- **BSalidaVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que caracterizan el envío de una respuesta a un órgano judicial, como por ejemplo aceptar solicitudes de préstamo de expedientes (tomos de asuntos o de libro), rechazar solicitudes de préstamo de expedientes, etc., y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende la clase BuzonVO.

- **UsuarioVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que caracterizan a los datos de un determinado usuario de la aplicación del Sistema de Archivo y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende a la clase *Usuario*.

  Estos objetos son empleados por la aplicación para comprobar que el usurario que accede a la misma es correcto, para dar de alta nuevos usuarios, etc.

- **RolVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que caracterizan el rol que tiene un determinado usuario de la aplicación del Sistema de Archivo y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende a la clase *DataVO* e implementa el interfaz *Rol*.

  Estos objetos son empleados por la aplicación para comprobar que permisos tiene un determinado usurario que accede a la misma, es decir que operaciones puede realizar un determinado usuario sobre la aplicación en función de lo que le permita su rol.

- **BloqueoVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que se emplean para establecer y comprobar cuando un determinado expediente del Sistema de Archivo esta bloqueado y que proporcionará/ recibirá un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende a la clase *DataVO.*

  Cuando un expediente (asunto o libro) este siendo editado por un usuario entonces se utilizará este objeto para establecer que dicho expediente está bloqueado y no puede ser editado por ningún otro usuario que intente acceder al mismo. Cuando un usuario intente acceder a los datos de un expediente deberá comprobar que no este bloqueado por otro.

- **NumeradorVO**, clase empleada por el cliente para crear aquellos objetos con los que se recibe/ envía aquellos atributos que nos permiten obtener/ mantener los identificadores internos que utilizamos en los expedientes (asuntos y libros), tomos, movimientos, operaciones de salida y cajas. Estos objetos serán proporcionados/ recibidos por un *EJB* de la lógica de negocios u objeto que se encargue del acceso a la base de datos. Esta clase extiende a la clase *DataVO*.

  Cada vez que se registre un nuevo expediente, se genere una operación de salida, se cree un nuevo tomo se emplearán estos objetos para solicita un identificador interno para cada objeto que se cree en la base de datos.

3.3.2 Data Access Objects

Para abstraer y encapsular todos los accesos a las fuentes de datos se ha seguido el patrón de diseño J2EE "*Data Access Object*".

De esta forma los objetos que se han creado siguiendo este patrón implementarán los mecanismos de acceso a la fuente de datos necesarios. Sea cual sea el mecanismo de almacenamiento de datos empleado, los objetos "*Data Access Object*" controlarán los accesos a las fuentes de datos ocultando a los clientes todos los detalles de implementación.

El patrón "*Data Access Object*" nos permitirá adaptar nuestra aplicación a diferentes mecanismos de almacenamiento sin que se vean afectados ni los clientes ni los componentes de la lógica de negocio. De esta forma, los cambios de implementación en el acceso a los datos solo afectarían a los objetos "*Data Access Object*".

3.3.2.1 Clases generales

Las clases que se describen en este punto, han sido utilizada para encapsular aquellos atributos y métodos comunes a algunas de las clases que implementan el patrón "Data Access Object", estas clases genéricas se describen a continuación.

- **MasterDAO**, clase abstracta genérica que se emplea para realizar el acceso a los datos. Esta clase ofrece la básica funcionalidad encargada de controlar la conexión a la base de datos, el control de transacciones, el procesado de sentencias SQL, etc.

- **DataDAO**, clase abstracta base de la que deberán extender todos los objetos que implementen el patrón "*Data Access Object*". Esta clase ofrece toda la funcionalidad necesaria para la generación dinámica de SQL.

La relación entre las clases "*MasterDAO*" y "*DataDAO*" se pueden apreciar en el siguiente diagrama de clases.

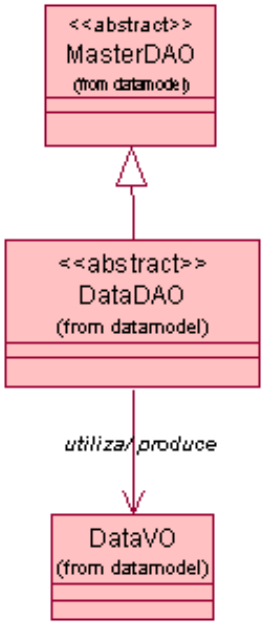

Figura 8. Diagrama de clases DataDAO

3.3.2.2 Clases Value Object

Las clases que se han creado siguiendo el patrón "*Data Access Object*", que extienden la clase "*DataDAO*", se especifican a continuación. Estas clases tienen asociadas unos ficheros XML que establecen la correspondencia entre el atributo de la clase VO y el

campo de la tabla que se le corresponde en la base de datos . Veamos un ejemplo de estos ficheros:

```
<?xml version="1.0" encoding="UTF-8"?>
<DAOConfiguration table="DEXPTOMO">
        <VOFields>
                <!-- Campos tabla DEXPTOMO -->
                <Field voName="codOrganoArchivo" fdName="CODORGARCH" tbName="DEXPTOMO"/>
                <Field voName="idExpediente"     fdName="IDEXP"      tbName="DEXPTOMO"/>
                <Field voName="idTomo"   fdName="IDEXPTOMO"          tbName="DEXPTOMO"/>
                <Field voName="codTomo"   fdName="CODTOMO"          tbName="DEXPTOMO"/>
                <Field voName="codOrganoActual"   fdName="CODORGACTUAL"
        tbName="DEXPTOMO"/>
                <Field voName="descripcion"                         fdName="DESCRIPCION"
        tbName="DEXPTOMO"/>
        </VOFields>
</DAOConfiguration>
```

Además en estos ficheros también se especifican las sentencias sql que se ejecutarán utilizando la clase, de la siguiente manera:

```
        <Query name="UPDATE_CAJA" value="UPDATE DCAJA SET estanteria = ? , balda = ?
        WHERE codorgarch = ? and idcaja = ? "/>
```

Podemos apreciar que se utiliza un signo de interrogación como marcador del lugar donde se agregarán los valores una vez ejecutada la consulta. Esto es debido a que se utiliza el objeto PreparedStatement[3].

Antes de procesar una consulta SQL, el SGBD tiene que compilarla. La compilación ocurre al llamar a cualquiera de los métodos de ejecución del objeto Statement. La compilación de una consulta implica un gasto aceptable si dicha consulta se llama solamente una vez. Sin embargo, el proceso de compilación puede ser costoso si la misma instancia del componente J2EE ejecuta la misma consulta varias en el transcurso de la misma sesión.

El objeto PreparedStatement permite compilar la consulta previamente a su ejecución, lo que mejorará los tiempos de acceso a la base de datos.

Las clases utilizadas en nuestra aplicación han sido las siguientes:

- **CajaDAO**, clase encargada de proporcionar el acceso a los datos característicos de las cajas y los datos que tengan alguna relación con las cajas que se encuentran en la base de datos. Esta se encargará de realizar todas aquellas consultas, inserciones, actualizaciones y borrado de datos relacionados con las cajas del Sistema de Archivo.

  Esta clase tiene un fichero XML asociado (*CajaDAO.xml*) que contiene todas las sentencias SQL que se pueden ejecutar utilizando esta clase.

- **TomoDAO**, clase encargada de proporcionar el acceso a los datos característicos de los tomos y los datos que tengan alguna relación con las tomos (identificador del expediente al que pertenece el tomo, etc) que se encuentran en la base de datos. Esta se encargará de realizar todas aquellas consultas, inserciones, actualizaciones y borrado de datos relacionados con los tomos del Sistema de Archivo.

  Esta clase tiene un fichero XML asociado (*TomoDAO.xml*) que contiene todas las sentencias SQL que se pueden ejecutar utilizando esta clase.

- **LibroDAO**, clase encargada de proporcionar el acceso a los datos característicos de los libros de registro que se encuentran en la base de datos. Esta se encargará de realizar todas aquellas consultas, inserciones, actualizaciones y borrado de datos relacionados con los libros del Sistema de Archivo.

  Esta clase tiene un fichero *XML* asociado (*LibroDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **AsuntoDAO**, clase encargada de proporcionar el acceso a los datos característicos de los asuntos que se encuentran en la base de datos. Esta clase se encargará de realizar todas aquellas consultas, inserciones, actualizaciones y borrado de datos relacionados con los asuntos del Sistema de Archivo.

  Esta clase tiene un fichero *XML* asociado (*AsuntoDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **OrganoDAO**, clase encargada de proporcionar el acceso a los datos característicos de los órganos judiciales se encuentran en la base de datos. Esta se encargará de realizar todas aquellas operaciones *SQL* relacionadas con los órganos que utilizan el Sistema de Archivo.

  Esta clase tiene un fichero *XML* asociado (*OrganoDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **IntervinienteDAO**, clase encargada de proporcionar el acceso a los datos característicos de los intervinientes y los datos que tengan alguna relación con estos (identificador del expediente al que esta asociado el interviniente, etc) que se encuentran en la base de datos. Esta clase se encargará de realizar todas aquellas consultas, inserciones, actualizaciones y borrado de intervinientes relacionados con un determinado expediente que se encuentra en el Sistema de Archivo.

  Esta clase tiene un fichero *XML* asociado (*IntervinienteDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **UsuarioDAO**, clase encargada de proporcionar el acceso a los datos característicos de un determinado usuario de la aplicación del Sistema de Archivo. Esta clase se encargará de realizar todas aquellas operaciones *SQL* relacionadas con los usuarios que utilizan el Sistema de Archivo, como obtener datos de un usuario, crear un nuevo usuario, etc.

Esta clase tiene un fichero *XML* asociado (*UsaurioDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **RolDAO**, clase encargada de proporcionar el acceso a los datos característicos del rol de un determinado usuario registrado en la aplicación del Sistema de Archivo. Esta clase se encargará de realizar todas aquellas operaciones *SQL* relacionadas con el rol de un usuario que utilice el Sistema de Archivo como obtener los permisos que tiene un determinado usuario en la aplicación, modificar los permisos del usuario, etc.

  Esta clase tiene un fichero *XML* asociado (*RolDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **MovimientoDAO**, clase encargada de proporcionar el acceso a los datos característicos de los movimientos que se han realizado sobre los tomos en los que se dividen los expedientes (asuntos y libros), tales como usuario que realiza el movimiento, tipo de movimiento, etc. Esta clase se encargará de realizar todas aquellas operaciones *SQL* relacionadas con los movimientos que se realizan sobre los tomos de los expedientes que hay, se registran o salen del Sistema de Archivo.

  Esta clase tiene un fichero *XML* asociado (*MovimientoDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **BEntradaDAO**, clase encargada de proporcionar el acceso a los datos característicos de las operaciones de entrada de la aplicación del Sistema de Archivo, registradas en la base de datos. Esta clase se encargará de realizar todas aquellas operaciones *SQL* relacionadas con las operaciones de entrada al Sistema de Archivo, tales como devoluciones de préstamos de expedientes, rechazo de envíos, etc.

Esta clase tiene un fichero *XML* asociado (*BEntradaDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **BSalidaDAO**, clase encargada de proporcionar el acceso a los datos característicos de las operaciones de salida de la aplicación del Sistema de Archivo, registradas en la base de datos. Esta clase se encargará de realizar todas aquellas operaciones *SQL* relacionadas con las operaciones de salida al Sistema de Archivo, tales como envío de rechazo o aceptación de solicitudes de préstamos, etc.

  Esta clase tiene un fichero *XML* asociado (*BSalidaDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **BloqueoDAO**, clase encargada de proporcionar el acceso a los datos de la base de datos, utilizados para determinar que expediente ha sido bloqueado y bloquear o desbloquear expedientes. Esta clase se encargará de realizar todas aquellas operaciones *SQL* relacionadas con las operaciones de bloqueo de expedientes que se encuentran en el Sistema de Archivo, tales como bloquear un expediente antes de editarlo, comprobar si un expediente esta bloqueado por otro usuario, desbloquear un expediente, etc.

  Esta clase tiene un fichero *XML* asociado (*BloqueoDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

- **NumeradorDAO**, clase encargada de proporcionar el acceso a los datos de la base de datos, utilizados para obtener cual es el siguiente identificador interno que se va a utilizar para identificar los objetos con los que trabajamos en la base de datos, como los identificadores de expedientes, tomos, cajas, etc. Esta clase se encargará de realizar todas aquellas operaciones *SQL* relacionadas con el mantenimiento de identificadores.

Esta clase tiene un fichero *XML* asociado (*NumeradorDAO.xml*) que contiene todas las sentencias *SQL* que se pueden ejecutar utilizando esta clase.

En el siguiente diagrama de clases se muestran las relaciones entre las diferentes clases e interfaces que hemos comentado, las clases que implementan el patrón "*Value Object* " y "*Data Object Access*", los interfaces que se emplean, etc.

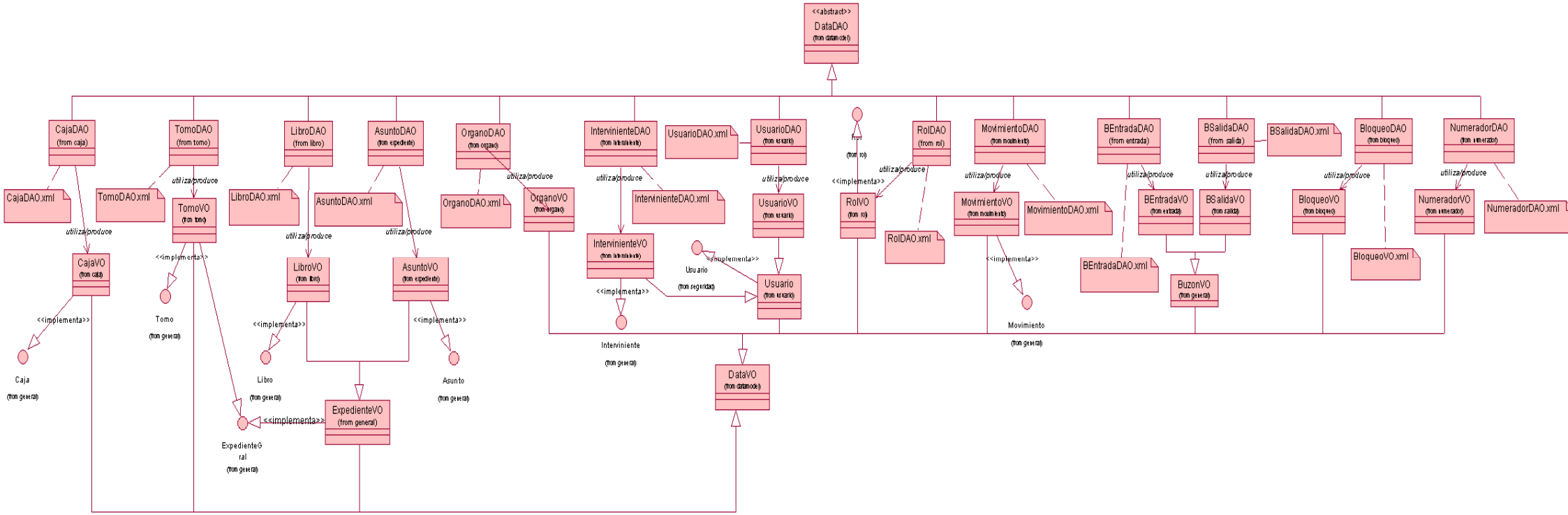

Figura 9. Diagrama de clases VO y DAO

3.3.3 Modelo de datos

La aplicación del Sistema de Archivo utiliza *EJB* de sesión, objetos cuyas clases implementan el patrón "*Data Access Object*", etc. Sin embargo, nuestra aplicación necesita acceder a datos que estarán compuestos de múltiples objetos, por ejemplo, es necesario acceder a un expediente con múltiples tomos o expedientes con múltiples intervinientes, etc.

El siguiente diagrama de clases nos muestra el modelo de datos que utilizará el cliente para trabajar con la información que obtendrá de la parte de la lógica de negocios.

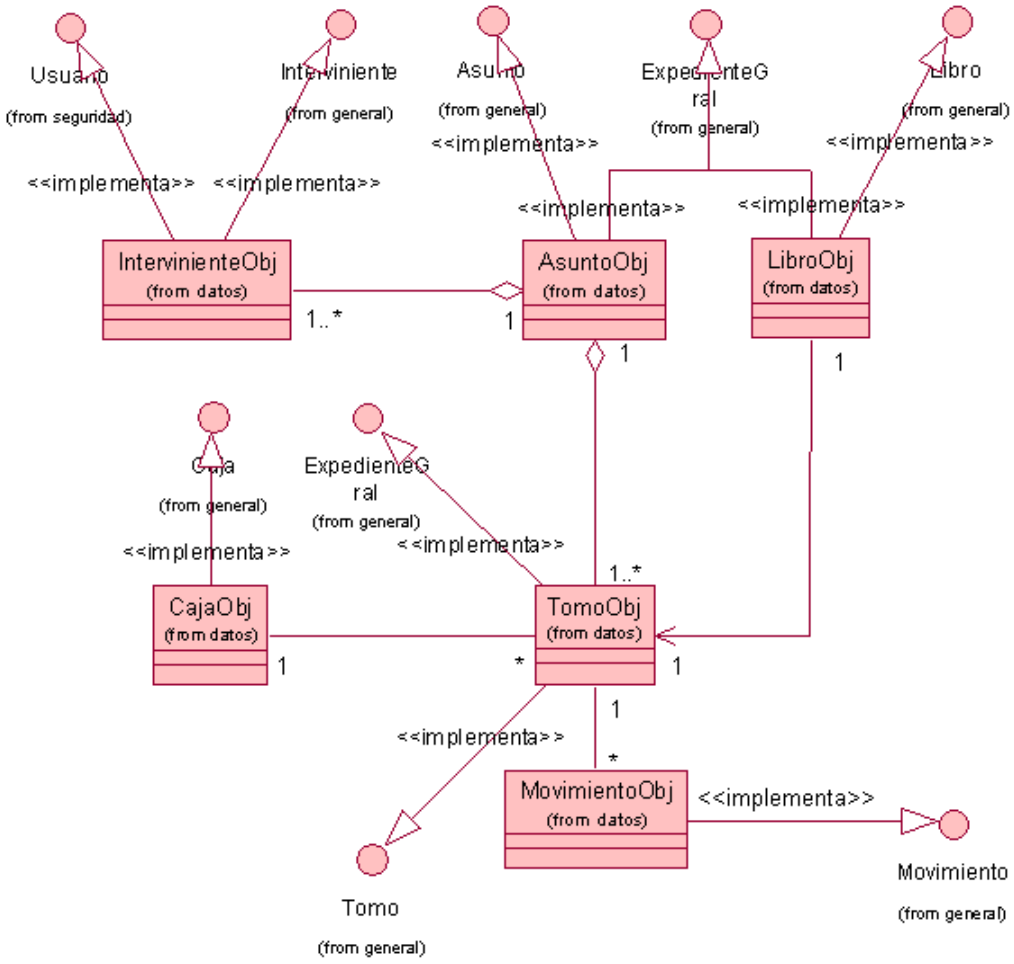

Figura 10. Diagrama de clases modelo de datos

Este modelo de datos nos permite trabajar con toda la información que se necesita de cualquier expediente y mantener todas las relaciones entre los diferentes objetos que la componen.

A continuación se describen todas las clases e interfaces que se han utilizado para el diseño del modelo de datos.

3.3.3.1 Interfaces

Los interfaces que se utilizan en el diseño del modelo de datos, son las mismas que se han utilizado en el diseño de las clases que implementan el patrón "*Value Object*.

La utilidad de estos interfaces reside en que, tanto las clases que implementen el patrón "*Value Object*", como las clases que implementen el modelo de objetos, deberán implementar el interfaz correspondiente, de esta forma se consigue que ambos clases compartan la misma signatura para los métodos y empleen el mismo tipo de datos, tal que cualquier cambio en los datos de los métodos supondrá un cambio en los métodos del interfaz y esto nos permitirá controlar los cambios que se tenga que realizar en las clases que extiendan de los interfaces.

3.3.3.2 Clases del modelo de datos

Las clases que se utilizan para construir el modelo de objeto expediente se describen a continuación.

- **AsuntoObj**, clase utilizada para trabajar con la información asociada un *asunto* de un determinado expediente. Un *asunto*, como podemos apreciar en el diagrama de clases, puede tener uno o varios *intervinientes* y estar desglosado en uno o varios *tomos*.

- **LibroObj**, clase utilizada para trabajar con la información de un *libro de registro* que está asociado a un expediente. Un *libro de registro* se utilizará como si fuese un *tomo*, por ello un *libro de registro* tiene un único *tomo*.

- **TomoObj**, clase utilizada para trabajar con la información de un *tomo* relacionado con un *asunto* o *libro de registro* que pertenece a un expediente determinado. Cada uno de los *tomos* tendrán una *caja* asociada en la que estarán almacenados y tendrán uno o varios *movimientos* asociados.

- **CajaObj**, clase utilizada para trabajar con la información de una *caja* que se encuentra en el Sistema de Archivo y que almacena uno o varios *tomos*, ya sean tomos correspondientes a *asuntos* o *libros de registro*.

- **IntervinienteObj**, clase utilizada para trabajar con la información de un *interviniente*. Un *interviniente* puede estar asociado a uno o varios *asuntos*.

- **MovimientoObj**, clase utilizada para trabajar con la información de un *movimiento* asociado a un *tomo*. La información de un movimiento nos describe que tipo de operación se ha realizado sobre un *tomo*, tales como prestar un *tomo*, enviar un *tomo*, registrar un *tomo*, etc.

3.4 LÓGICA DE NEGOCIO

En este apartado se describen todas aquellas clases que se han desarrollado que se encargarán de controlar todas las operaciones que tengan que ver con la lógica de negocio como los *EJB's* de sesión que se han creado, los mecanismos empleados para acceder a los *EJB's* y el resto de las clases que se utilizan.

3.4.1 Clases generales

Para realizar el diseño de los *EJB's* de sesión se ha seguido el mismo esquema que se ha empleado en el diseño del framework de SEINTEX y que veremos en los siguientes puntos.

3.4.1.1 Interfaces EJB remotos y locales

Los EJB's de sesión diseñados para la aplicación del Sistema de Archivo serán duales al igual que en el framework de SEINTEX, es decir, los clientes de la aplicación podrán acceder a ellos vía remota (cliente y EJB's se encuentran en contenedores diferentes) o local (cliente y EJB's se encuentran en el mismo contenedor).

Un contenedor de EJB [3] es una entidad proporcionada por un proveedor que se encuentra dentro del servidor de EJB, el cual gestiona los servicios de sistema para los EJB. El contenedor de EJB es uno de varios contenedores, cada uno de los cuales soporta un componente J"EE como son los servlets y JSP.

Un contenedor de EJB es una entidad proporciona un pool reutilizable de componentes distribuidos. Cada EJB se tiene que instalar en un contenedor de EJB. Por lo general, un contenedor contiene muchos EJB. Puede también haber varios contenedores de EJB. Sin embargo, un EJB se tiene que instalar solamente en un contenedor.

La petición de servicio del cliente llega al servidor web, el cual reenvía la petición del cliente al servidor adecuado. Por lo general, quien recibe la petición del cliente es el

servidorde servlets o de JSP, dependiendo del tipo de componente que se utilice para la aplicación J2EE.

El servlet o JSP utiliza JNDI para buscar el EJB. La búsqueda devuelve el objeto home del EJB, el cual se utiliza para cerrar el bean y devolver el objeto EJB. El acceso al Bean se hace a través de la interfaz remota del EJB.

Para ello los EJB's de sesión que se han diseñado deberán definir los interfaces que nos permiten acceder a los EJB's de forma remota y local. Las clases e interfaces que corresponden a cada EJB se definen en un el **descriptor de despliegue**, que es un archivo con sintaxis XML que se incluye dentro del Jar. Websphere posee un editor gráfico para este tipo de archivos, donde además podemos definir referencias a otros recursos externos como orígenes de datos u otros EJB's. Veamos un fragmento del descriptor de despliegue utilizado en la aplicación. En concreto el del EJB incluido en la plataforma para la gestión de los numeradores.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar id="ejb-jar_ID">
	<display-name>Numeradores</display-name>
	<enterprise-beans>
		<session id="NumeradorLocal">
			<ejb-name>NumeradorLocal</ejb-name>
			<local-home>mju.ntj.plataforma.ext13.numeradores.NumeradorLocalHome</local-home>
			<local>mju.ntj.plataforma.ext13.numeradores.NumeradorLocal</local>
			<ejb-class>mju.ntj.plataforma.ext13.numeradores.NumeradorBean</ejb-class>
			<session-type>Stateless</session-type>
			<transaction-type>Bean</transaction-type>
			<env-entry>
				<description>Cadena con la direccion del recurso JNDI</description>
				<env-entry-name>dataSourceJNDI</env-entry-name>
				<env-entry-type>java.lang.String</env-entry-type>
				<env-entry-value>java:comp/env/jdbc/desaweb</env-entry-value>
			</env-entry>
			<resource-ref id="ResourceRef_1149071386281">
				<description></description>
				<res-ref-name>jdbc/desaweb</res-ref-name>
				<res-type>javax.sql.DataSource</res-type>
				<res-auth>Container</res-auth>
				<res-sharing-scope>Shareable</res-sharing-scope>
			</resource-ref>
		</session>
		<session id="NumeradorRemote">
```

```
                        <ejb-name>NumeradorRemote</ejb-name>

        <home>mju.ntj.plataforma.ext13.numeradores.NumeradorRemoteHome</home>
                        <remote>mju.ntj.plataforma.ext13.numeradores.NumeradorRemote</remote>
                        <ejb-class>mju.ntj.plataforma.ext13.numeradores.NumeradorBean</ejb-class>
                        <session-type>Stateless</session-type>
                        <transaction-type>Container</transaction-type>
                        <resource-ref>
                                <description></description>
                                <res-ref-name>jdbc/numerador</res-ref-name>
                                <res-type>javax.sql.DataSource</res-type>
                                <res-auth>Container</res-auth>
                                <res-sharing-scope>Shareable</res-sharing-scope>
                        </resource-ref>
                </session> -->
        </enterprise-beans>
</ejb-jar>
```

Los interfaces remotos extenderán el interfaz *javax.ejb.EJBObject* así como a cada uno de los interfaces *<Nombre>Service* que les corresponda. Este interfaz estará vacío ya que la declaración de los métodos del interfaz ya vendrá especificada en los interfaces *<Nombre>Service*, además, será en estos interfaces donde se defina también las excepciones que se lanzará.

Los interfaces locales extenderán el interfaz *javax.ejb.EJBLocalObject* y, al igual que los interfaces remotos, extenderán a cada uno de los interfaces *<Nombre>Service* que les corresponda. En este caso los interfaces locales sí que deberán declarar los métodos que implementará la clase que los extienda ya que esta variará levemente en el sentido de que no lanzarán excepciones remotas.

Los interfaces *<Nombre>Service* de los que extienden los interfaces remotos y locales, extenderán al interfaz *BusinessService*, definirán los métodos de la lógica de negocios que implemente cada *EJB* de sesión y estos métodos deberán lanzar excepciones de la clase *java.rmi.RemoteException* y de la clase *seintex.base.datamodel.DataModelError*, clase que se encuentra desarrollada en el framework de SEINTEX en el *JAR* SeintexBaseDAO.

Por cada uno de los interfaces *<Nombre>Service* que creemos, tendremos un clase que extiende la clase *seintex.base.datamodel.DataModelError* para lanzar excepciones cuando estemos trabajando con EJB's de sesión en local.

Los interfaces que nos permitirán crear un *EJB* de forma remota se muestran en la siguiente figura, estos interfaces extenderán el interfaz *javax.ejb.EJBHome*, lanzarán excepciones de la clase *javax.rmi.CreateException* y *javax.rmi.RemoteException* y deberán tener definido al menos un método **create** que devolverá el interfaz remoto correspondiente que extiende al interfaz *javax.ejb.EJBObject*.

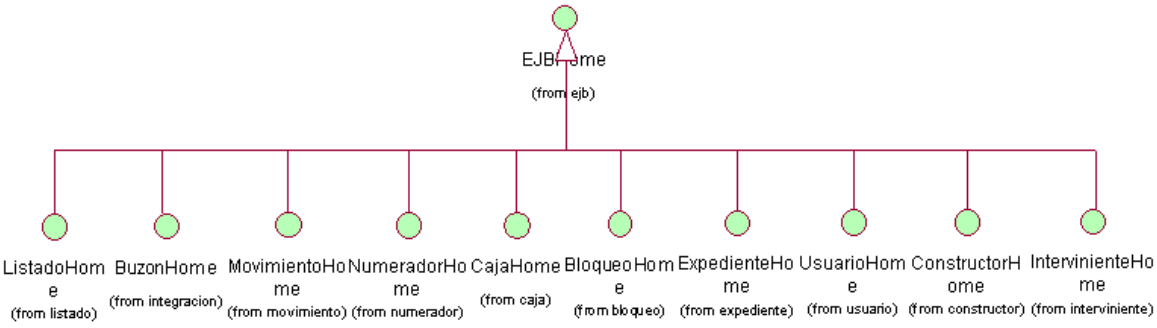

Figura 11. Diagrama de clases interfaces EJBHome acceso remoto

Los interfaces que nos permitirán crear un *EJB* de forma local se muestran en la siguiente figura, estos interfaces extenderán el interfaz *javax.ejb.EJBLocalHome*, lanzará excepciones de la clase *javax.rmi.CreateException*, En este caso no se lanzarán excepciones remotas ya que el *EJB* esta funcionando en el mismo contenedor que el cliente, y deberán tener definido al menos un método **create** que devolverá el interfaz local correspondiente que extiende al interfaz *javax.ejb.EJBLocalObject*.

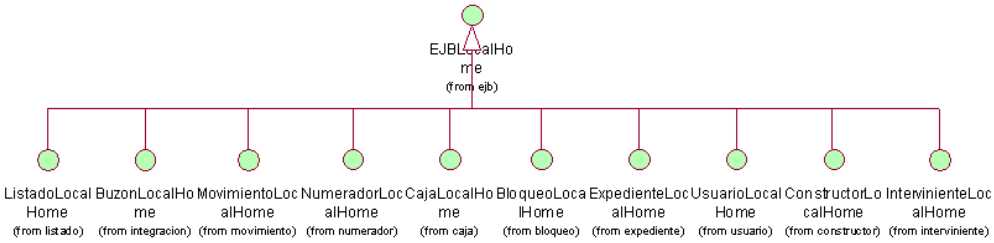

Figura 12. Diagrama de clases interfaces EJBLocalHome acceso remoto

3.4.1.2 EJB BusinessBean

En el siguiente diagrama se muestra el *EJB* de sesión general el cual extenderán todas las clases que desarrollemos para obtener los *EJB's* de sesión que diseñemos.

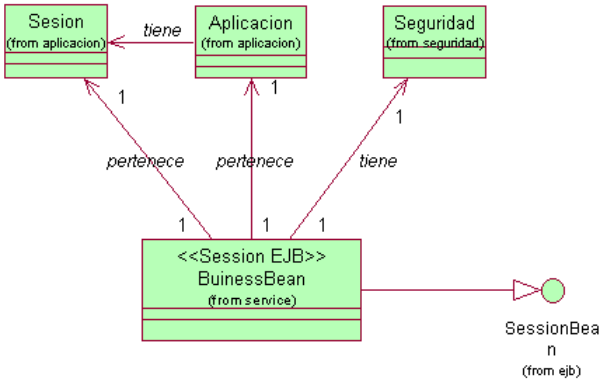

Figura 13. Diagrama de clases del EJB de sesión BusinessBean

Este EJB general, al implementar el interfaz **SessionBean**, implementa los métodos *ejbCreate*, *ejbPassivate*, *ejbActivate* y *ejbRemove*, que todo EJB debe implementar, aporta un *log* y métodos de lectura del contexto de sesión, proporcionara métodos para acceder al entorno de aplicación, de sesión y de seguridad, es decir, los métodos y atributos básicos que emplearán el resto de *EJB's* que se desarrollen para la aplicación de Sistema de Archivo. Estas clases están encapsuladas en los siguientes *JAR's* dentro del framework de SEINTEX, la clase *BusinessBean*, *Sesion* y *Aplicacion* se encuentra en el *JAR* **SeintexBaseEJB**, dentro de los paquetes *seintex.base.service* y *seintex.base.aplicación*, y la clase *Seguridad* se encuentra en el *JAR* **SeintexBaseSEG**, dentro del paquete *seintex.base.seguridad*.

3.4.1.3 BusinessServiceFacade y BusinessServiceLocator

Para acceder a los diferentes EJB's de sesión que hemos diseñado se utilizan las clases que **BusinessServiceFacade** y **BusinessServiceLocator**, que implementan los patrones "*SessionFaçade*" y "*ServiceLocator*" respectivamente.

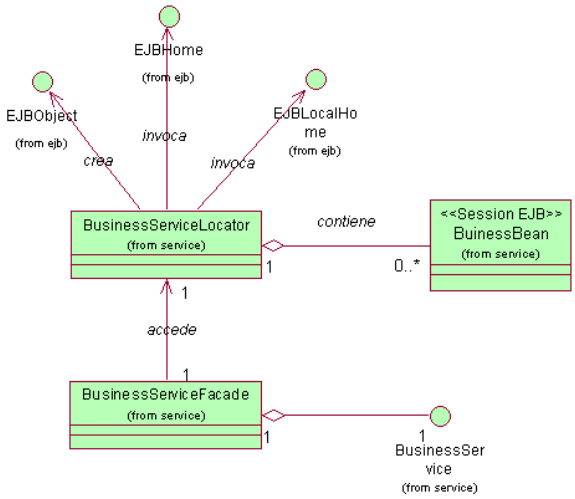

Figura 14. Diagrama de clases del BusinessService Facade y BusinessServiceLocator

Como se puede apreciar en el diagrama de clases anterior la clase BusinessServiceFacade se emplea para abstraer el acceso a los diferentes *EJB's* de sesión que diseñemos, tal que nos permitirá trabajar con cualquier *EJB's* que hemos diseñado de una forma uniforme. El *BusinessServiceFacade* accederá al *BusinessServiceLocator* (cuya utilidad se describe a continuación) para obtener un *EJB's* de sesión (que como veremos más adelante implementa el interfaz *BusinessService*) y del que mantendrá una referencia para trabajar con el.

La clase BusinessServiceLocator mantiene un mapa de objetos de la clase *BusinessBean*, tanto locales como remotos (*EJBLocalHome* o *EJBHome*). Esta clase será la encargada de devolver a la clase *BusinessServiceFacade* aquellos *EJB's* de sesión, locales o remotos, que le soliciten. Si el *EJB* de sesión solicitado ya existe, la clase *BusinessServiceLocator* le devolverá una referencia a este y en caso de no existir lo crearía, devolvería una referencia al objeto y guardaría otra referencia en el mapa de *EJB's*.

Por lo tanto, como se verá en los siguientes apartados, para acceder a los diferentes *EJB's* de sesión que tenemos, se han diseñado clases que extienden la clase *BusinessServiceFacade*. De esta forma tendremos acceso al *EJB* que solicitamos y a sus métodos. Estas clases se encuentran desarrolladas en el framework de SEINTEX, dentro del *JAR* seintexBaseEJB.

3.4.2 EJB's de sesión

En este apartado se describen los diferentes *EJB's* de sesión que han sido diseñados, que clases se emplean para desarrollar los EJB's de sesión y cuál es su finalidad.

- **ExpedienteBean**, *EJB* de sesión que implementa toda la lógica de negocios relacionada con el control y mantenimiento de expedientes. Se encargaría de realizar operaciones de edición de expedientes, registro de expedientes, consulta de expedientes, etc. Para realizar ciertas operaciones que implementa el *ExpedienteBean*, como por ejemplo editar un expediente, este necesita utilizar otros *EJB's* de sesión, el *EJB BloqueoBean* a parte del *EJB ExpedienteBean*, que se describen a continuación.

- **CajaBean**, *EJB* de sesión que extiende el *EJB BusinessBean* y que implementa toda la lógica de negocios relacionada con mantenimiento de cajas en el Sistema de Archivo. Se encarga de realizar las operaciones de crear nuevas cajas, realizar consultas sobre cajas, eliminar cajas, etc

- **IntervinienteBean**, *EJB* de sesión que extiende el *EJB BusinessBean* y que implementa toda la lógica de negocios relacionada con el control y mantenimiento de intervinientes. Se encarga de realizar las operaciones de agregar nuevos intervinientes a un expediente, eliminar intervinientes de un expediente, consultar intervinientes, etc.

- **MovimientoBean**, , *EJB* de sesión que extiende el *EJB BusinessBean* y que implementa toda la lógica de negocios relacionada con el control de movimientos que se realizan sobre los expedientes. Se encarga de realizar las operaciones de consulta de movimientos que se han realizado sobre un expediente, crear nuevos movimientos para un expediente, etc.

- **BuzonBean**, *EJB* de sesión que extiende el *EJB BusinessBean* y que implementa toda la lógica de negocios relacionada con el control de los buzones de entrada y de salida a la aplicación de Sistema de Archivo. Se encarga de realizar las operaciones de recepción de solicitudes de préstamos, recepción de devoluciones de expedientes, registro de nuevos expedientes, envío de rechazo/ aceptación de solicitudes, consulta de los buzones de entrada y salida, etc.

- **ListadoBean**, , *EJB* de sesión que extiende el *EJB BusinessBean* y que implementa toda la lógica de negocios relacionada con la obtención de todo tipo de listados, calculo de estadísticas y emisión de etiquetas. Se encarga de realizar operaciones de consulta sobre la base de datos para obtener la información que necesita para construir los listados que se le solicitan.

- **ConstructorBean**, , *EJB* de sesión que extiende el *EJB BusinessBean* y que implementa toda la lógica de negocios relacionada con la construcción del modelo de datos que ha de ser devuelto al usuario a partir de los objetos "*Value Object*" que recibe.

- **UsuarioBean**, , *EJB* de sesión que extiende el *EJB BusinessBean* y que implementa toda la lógica de negocios relacionada con el control y mantenimiento de los usuarios que utilizan la aplicación de Sistema de Archivo. Se encarga de realizar las operaciones de registro de nuevos usuarios, dar permisos a usuarios, restringir permisos, etc.

- **BloqueoBean**, , *EJB* de sesión que extiende el *EJB BusinessBean* y que implementa toda la lógica de negocios relacionada con el control del bloqueo de expedientes. Se encarga de realizar las operaciones de bloqueo de aquellos expedientes que están siendo editados por un determinado usuario, desbloquearlos cuando termine la edición, etc.

- **NumeradorBean**, , *EJB* de sesión que extiende el *EJB BusinessBean* y que implementa toda la lógica de negocios relacionada con el control de los

numeradores internos que se utilizan para identificar internamente los expedientes (asuntos y libros), tomos, movimientos, etc. Se encarga de realizar las operaciones que nos devuelvan y actualicen estos identificadores.

El siguiente diagrama de clases nos muestra todos los EJB's de sesión que han sido diseñados para la aplicación del sistema de archivo.

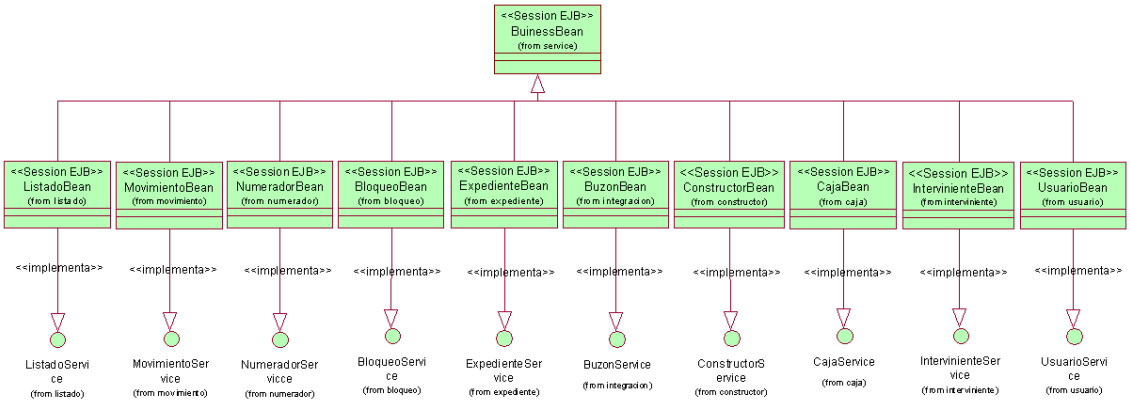

Figura 15. Diagrama de clases de los EJB's de sesión diseñados

Como se puede apreciar en el diagrama de clases anterior todos los *EJB's* de sesión diseñados extienden la clase *BusinessBean*, que como hemos indicado anteriormente extiende a la clase *SessionBean*, con lo cual heredan los métodos de los *EJB's* de sesión y aparte cada uno implementa su interfaz *<Nombre>Service* correspondiente donde están definidos los métodos que deberán implementar los *EJB's*.

3.4.3 Controladores de acceso a los servicios

Para poder acceder a los *EJB's* de sesión que hemos especificado se utiliza, como hemos descrito anteriormente, una clase diferente para cada *EJB* de sesión que hemos desarrollado que extiende la clase *BusinessServiceFacade*, encargada de utilizar la clase *singleton BusinessServiceLocator* para obtener el *EJB* de sesión que se solicita y proporcionar una referencia al mismo para poder utilizarlo.

- **ExpedienteServiceManager**, controlador del servicio para acceder al *EJB* de sesión *ExpedienteBean.* Se encargará de obtener una referencia al *EJB*

ExpedienteBean e invocar a los servicios de este que el cliente necesite utilizar. Por ello, en el *ExpedienteServiceManager*, tendremos los mismos métodos que se han definido en el interfaz *ExpedienteService*.

- **MovimientoServiceManager**, controlador del servicio para acceder al *EJB* de sesión *MovimientoBean*. Se encargará de obtener una referencia al *EJB MovimientoBean* e invocar a los servicios de este que el cliente necesite utilizar. Por ello, en el *MovimientoServiceManager*, tendremos los mismos métodos, que se han definido en el interfaz *MovimientoService*.

- **IntervinienteServiceManager**, controlador del servicio para acceder al *EJB* de sesión *IntervinienteBean*, como su nombre indica se encargará de obtener una referencia al *EJB IntervinienteBean* e invocar a los servicios de este que el cliente necesite utilizar. Por ello, en el *IntervinienteServiceManager*, tendremos los mismos métodos que se han definido en el interfaz *IntervinienteService*.

- **CajaServiceManager**, controlador del servicio para acceder al *EJB* de sesión *CajaBean*, se encargará de obtener una referencia al *EJB CajaBean* e invocar a los servicios de este, que el cliente necesite utilizar. En el *CajaServiceManager* tendremos los mismos métodos que tenemos definidos en el interfaz *CajaService*.

- **ListadoServiceManager**, controlador del servicio para acceder al *EJB* de sesión *ListadoBean*. Es el encargado de obtener una referencia al *EJB ListadoBean* e invocar a los servicios de este. Por ello, en el *ListadoServiceManager* tendremos los mismos métodos que tenemos definidos en el interfaz *ListadoService*.

- **UsuarioServiceManager**, controlador del servicio para acceder al *EJB* de sesión *UsuaurioBean*. Es el encargado de obtener una referencia al *EJB UsuarioBean* e invocar a los servicios que el cliente necesite utilizar. En el

UsuarioServiceManager tendremos los mismos métodos que tenemos en el *UsuarioService*.

- **BuzonServiceManager**, controlador del servicio para acceder al *EJB* de sesión *BuzonBean*. Se encargará de obtener una referencia al *BuzonBean* e invocar a los servicios de este. En el *BuzonServiceManager* tendremos los mismos métodos que tenemos en el interfaz *BuzonService*.

- **BloqueoServiceManager**, controlador del servicio para acceder al *EJB* de sesión Bloqueo*Bean*. Se encargará de obtener una referencia al *EJB BloqueoBean* e invocar a los servicios de este que el cliente necesite utilizar, bloquear un determinado expediente que esta siendo editado, desbloquear un expediente cuya edición ha finalizado, etc. En el *BloqueoServiceManager*, tendremos los mismos métodos que han sido definidos en el interfaz *BloqueoService*.

- **NumeradorServiceManager**, controlador del servicio para acceder al *EJB* de sesión *NumeradoBean*. Se encargará de obtener una referencia al *EJB NumeradorBean* e invocar a los servicios de este que el cliente necesite utilizar, como obtener un determinado identificador para un movimiento, expediente, etc. En el *NumeradorServiceManager* tendremos los mismos métodos que han sido definidos en el interfaz *NumeradorService*.

- **ConstructorServiceManager**, controlador del servicio para acceder al EJB de sesión ConstructorBean. Se encargará de obtener una referencia al EJB ConstructorBean e invocar a los servicios de este, que el cliente necesite utilizar. En el ConstructorServiceManager tendremos los mismos métodos que han sido definidos en el interfaz ConstructorService.

En el siguiente diagrama de clases se puede apreciar cómo se relacionan los diferentes controladores de acceso a los servicios con cada uno de los EJB's de sesión que hemos desarrollado.

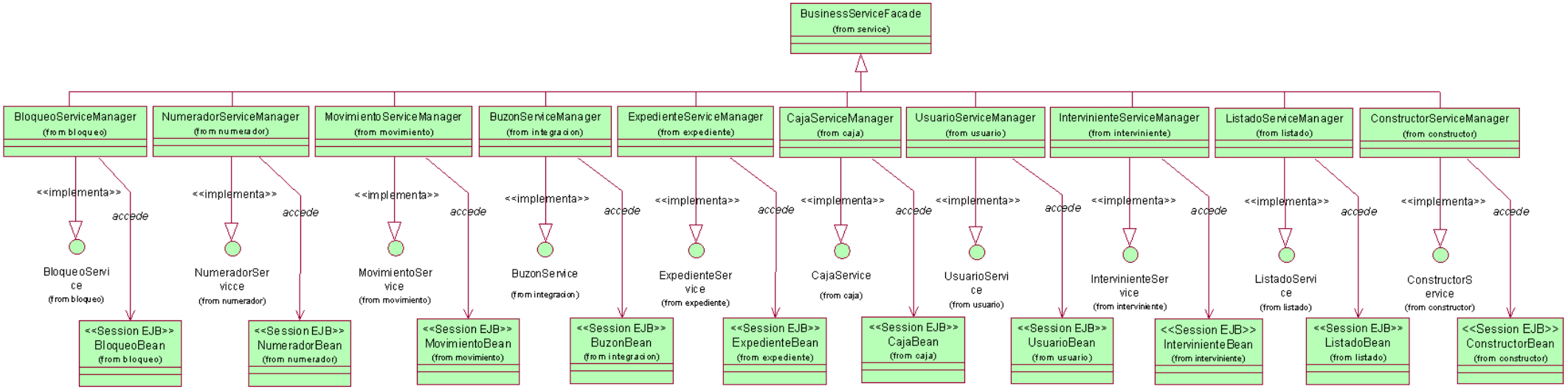

Figura 16. Diagrama de clases de los controladores de acceso a los EJB's de sesión diseñados

3.4.4 Interacción entre los diferentes EJB's de sesión

En algunos casos es necesario que algunos *EJB's* de sesión tenga acceso a otros para poder realizar ciertas operaciones que estos no pueden realizar directamente o no resulta conveniente que las realicen para mantener separada la funcionalidad de cada uno de los *EJB's* que han sido diseñados.

En este apartado se mostrará que *EJB's* de sesión utilizan las correspondientes clases de control de acceso a los *EJB's* para utilizar sus funcionalidades y realizar ciertas operaciones.

- **EJB ExpedienteBean**, el siguiente diagrama de clases nos permite ver a que otros *EJB's* de sesión podrá acceder el *EJB ExpedienteBean* para realizar todas aquellas operaciones que necesite realizar utilizando estos *EJB's*.

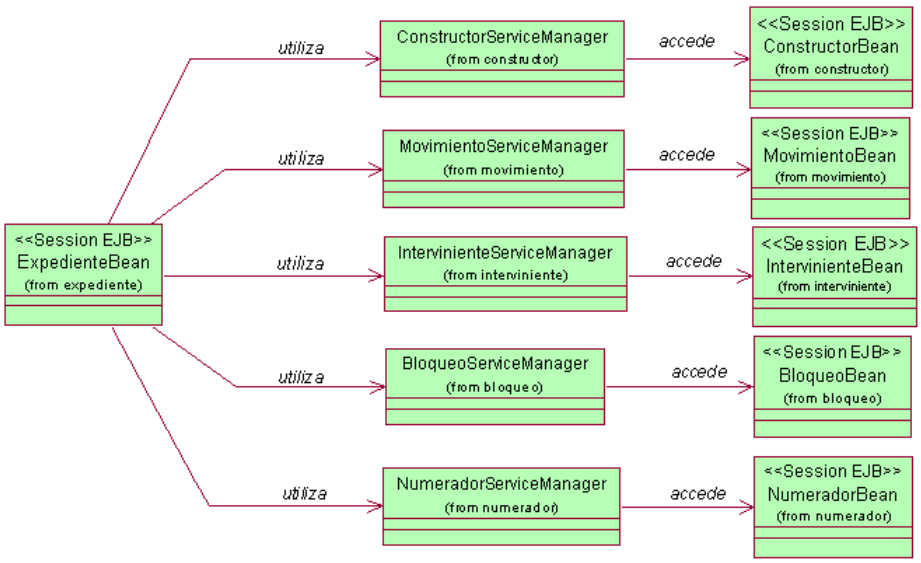

Figura 17. Diagrama de clases acceso EJB ExpedienteBean

Como podemos apreciar el EJB ExpedienteBean tiene acceso a los siguientes controladores de acceso para acceder a sus correspondientes EJB's de sesión.

– **ConstructorServiceManager**, controlador de acceso al *EJB ConstructorBean* que nos permitirá crear el modelo de datos que se le pasará al usuario, que más nos interese en función de los datos que

contienen los objetos que son de aquellas clases que implementen el patrón "*Value Object*".

- **MovimientoServiceManager**, controlador de acceso al *EJB MovimientoBean* que nos permitirá manejar las operaciones de movimientos que estén relacionados con un determinado expediente. De esta forma será posible trabajar con los movimientos de un expediente accediendo desde el *EJB ExpedienteBean*, antes de devolver algún resultado al cliente que solicito la operación.
- **IntervinienteServiceManager**, controlador de acceso al *EJB IntervinienteBean* que nos permitirá manejar los datos correspondientes a los intervinientes que estén asociados a un determinado expediente. De esta forma es posible realizar operaciones de añadir, actualizar o eliminar intervinientes desde el *EJB ExpedienteBean*, que tiene las operaciones para manipular cualquier expediente.
- **BloqueoServiceManager**, controlador de acceso al *EJB BloqueoBean* que nos permitirá realizar el bloqueo de un expediente desde el *EJB ExpedienteBean* cuando se estén editando los datos de un determinado expediente.
- **NumeradorServiceManager**, controlador de acceso al *EJB NumeradorBean* que nos permitirá obtener un identificador único desde el *EJB ExpedienteBean* cuando estemos realizando la operación de registrar un expediente nuevo.

- **EJB ListadoBean**, el siguiente diagrama de clases nos permite ver a que otros *EJB's* de sesión podrá acceder el *EJB ListadoBean*.

Figura 18. Diagrama de clases acceso EJB BuzonBean

En este caso el *EJB ListadoBean* empleará el controlador de acceso *ConstructorServiceManager* para acceder al *ConstructorBean* y poder construir

objetos complejos, que devolverán los datos que el usuario solicito para obtener un determinado listado.

- **EJB MovimientoBean**, el siguiente diagrama de clases nos permite ver a que otros *EJB's* de sesión podrá acceder el *EJB MovimientoBean*.

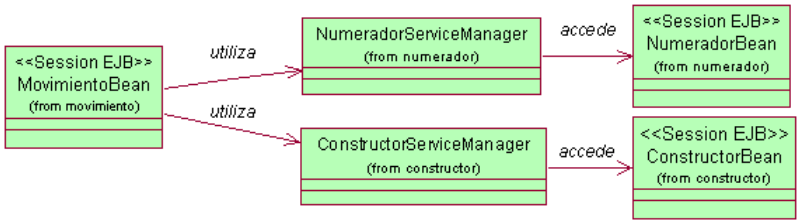

Figura 19. Diagrama de clases acceso EJB MovimientoBean

Como podemos apreciar el *EJB MovimientoBean* tiene acceso a los siguientes controladores de acceso para acceder a sus correspondientes *EJB's* de sesión.

- **ConstructorServiceManager**, el *EJB MovimientoBean* empleará el controlador de acceso *ConstructorServiceManager* para acceder al *ConstructorBean* y poder construir objetos complejos, que devolverán los datos que el usuario solicito para obtener una determinada consulta que comprenda los movimientos asociados para un determinado expediente, tomos de un expediente, etc.
- **NumeradorServiceManager**, el *EJB MovimientoBean* utilizará este controlador de acceso para utilizar los métodos de *EJB NumeradorBean* y poder obtener un identificador único para cada movimiento que se realice y se guarde en la base de datos.

- **EJB CajaBean**, el siguiente diagrama de clases nos permite ver a que otros *EJB's* de sesión podrá acceder el *EJB CajaBean*.

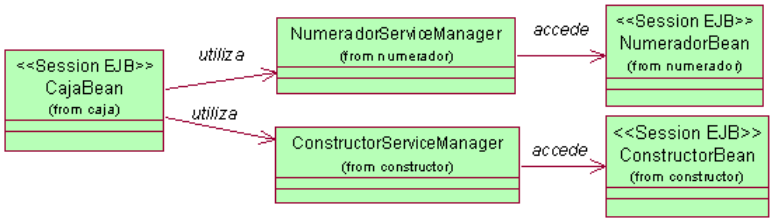

Figura 20. Diagrama de clases acceso EJB CajaBean

Como podemos apreciar el *EJB CajaBean* tiene acceso a los siguientes controladores de acceso para acceder a sus correspondientes *EJB's* de sesión.

- **ConstructorServiceManager**, el *EJB CajaBean* empleará el controlador de acceso *ConstructorServiceManager* para acceder al *ConstructorBean* y poder construir objetos complejos, que devolverán los datos que el usuario solicito para realizar una determinada consulta que comprenda los expedientes o tomos que hay en una determinada caja, etc.
- **NumeradorServiceManager**, el *EJB CajaBean* utilizará este controlador de acceso para utilizar los métodos de *EJB NumeradorBean* y poder obtener un identificador único para cada caja que se cree en el sistema de archivo.

- **EJB BuzonBean**, el siguiente diagrama de clases nos permite ver a que otros *EJB's* de sesión podrá acceder el *EJB BuzonBean*.

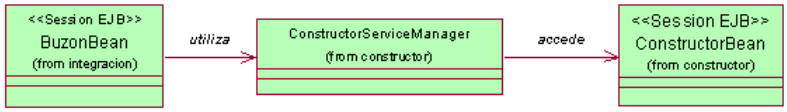

Figura 21. Diagrama de clases acceso EJB BuzonBean

Como podemos apreciar el *EJB BuzonBean* tiene acceso al controlador de acceso *NumeradorServiceManager* para utilizar los métodos de *EJB NumeradorBean* y poder obtener un identificador único para cada operación que se realice en los buzones de entrada o salida.

3.5 RELACIÓN ENTRE LAS DIFERENTES CAPAS

En los siguientes apartados se mostrarán como interactúan las clases que hemos creado para las diferentes capas de lógica de presentación, lógica de negocio y lógica de datos.

3.5.1 Relación entre la lógica de presentación y la lógica de negocio

En este apartado se muestra la relación que hay entre las clases que han sido diseñadas en la lógica de presentación y la lógica de sesión. Para ello veremos una serie de diagramas de clases que nos muestran como desde las clases de la lógica de presentación se accede a las clases de la lógica de negocio, que controlan los accesos a los *EJB's* de sesión, para realizar las operaciones que el usuario solicite.

3.5.1.1 Acciones sobre expedientes

Este apartado relaciona las clases de presentación encargadas de recibir las solicitudes de operaciones de los usuarios sobre los expedientes (asuntos, libros, tomos, intervinientes, etc.), con las clases de la lógica de negocio encargadas de realizar las operaciones sobre estos expedientes, ingresar expedientes, añadir interviniente, realizar movimientos de expedientes, etc.

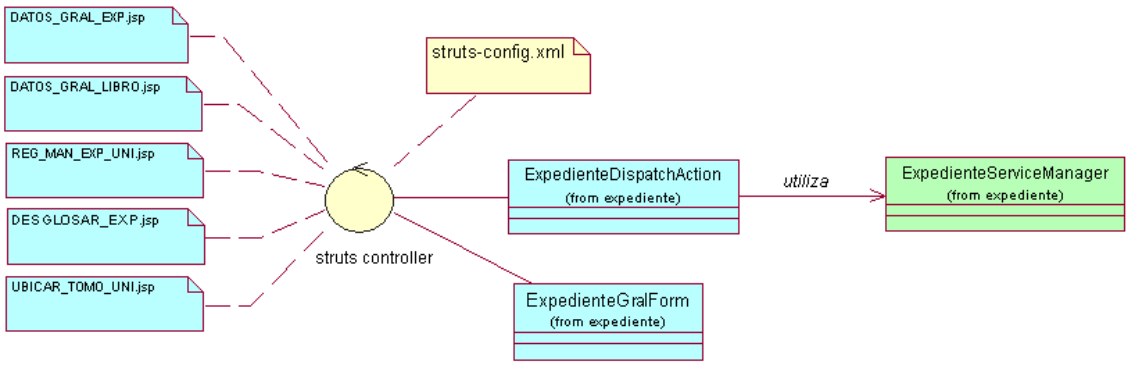

Figura 22. Relación clases expedientes lógica presentación con negocio

En el diagrama de clases anterior, podemos apreciar todas las acciones que se realicen sobre los expedientes, ya sean operaciones de edición (añadir intervinientes,

movimientos sobre los expedientes), inserción, etc., que un usuario realiza a través de un *JSP*, son recibidas por la clase de la lógica de presentación *ExpedienteDispatchAction*, la cual utilizará una referencia al *ExpedienteServiceManager* para solicitar las operaciones que necesite al *EJB ExpedienteBean,* que será el encargado de ejecutar las operaciones necesarias o bien utilizar otros controladores de acceso a otros *EJB's* para realizar otras operaciones si es necesario.

3.5.1.2 Acciones sobre listados

Este apartado relaciona las clases de presentación encargadas de recibir las solicitudes de los usuarios para realizar listados (listado de asuntos, de solicitudes de préstamo, de etiquetas de cajas, etc.), con las clases de la lógica de negocio encargadas de realizar estas operaciones.

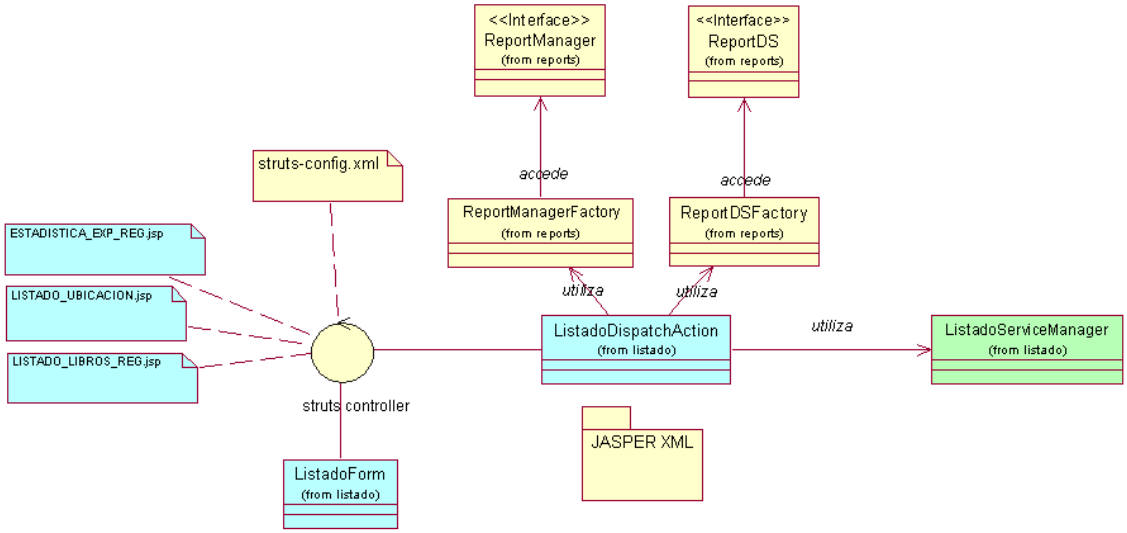

Figura 23. Relación clases listado lógica presentación con negocio

En este diagrama de clases, todas las acciones que requieran realizar un determinado listado por parte del usuario son recibidas por la clase de la lógica de presentación *ListadoDispatchAction*, la cual empleando una referencia a la clase *ListadoServiceManager* accederá al *EJB ListadoBean* el cual accederá a la base de datos empleando las clases que implementan el patrón "Data Access Object" para obtener los

datos que necesita para realizar el listado que le han solicitado. Estos datos serán devueltos a la clase ListadoDispatchAction, que utilizará las clases *ReportMangerFactory* y *ReportDSFactory* para realizar un listado que muestre lo que el usuario ha solicitado.

3.5.1.3 Acciones sobre cajas

Este apartado relaciona las clases de presentación encargadas de recibir las solicitudes de operaciones de los usuarios sobre las cajas que hay en el sistema de archivo (crear nuevas cajas, reubicar cajas, etc.), con las clases de la lógica de negocio encargadas de realizar dichas.

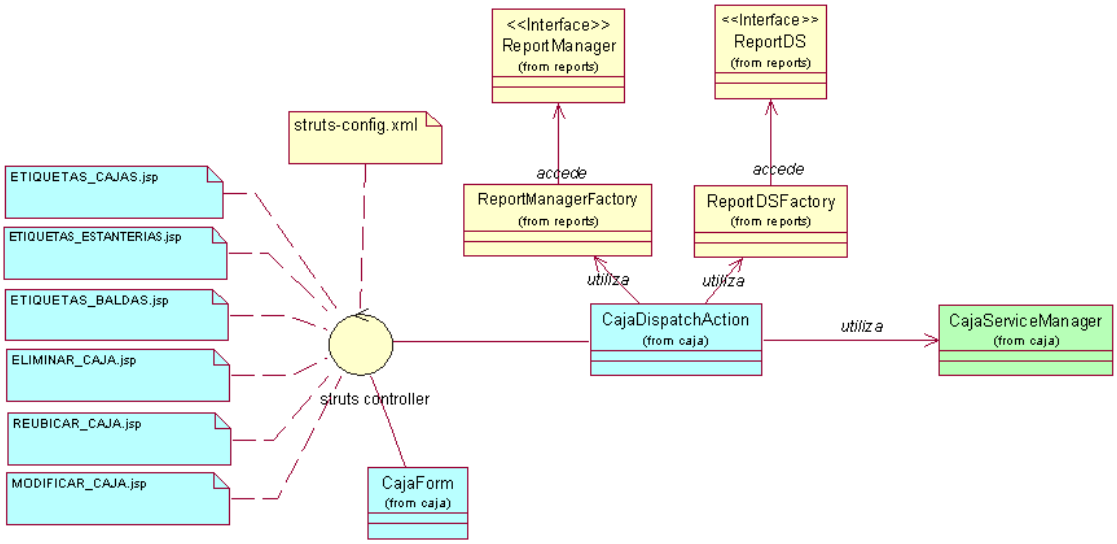

Figura 24. Relación clases cajas lógica presentación con negocio

Como podemos apreciar en el diagrama de clases, todas las acciones que se realicen sobre el mantenimiento de cajas que un usuario invoca utilizando el *JSP* correspondiente son recibidas por la clase de la lógica de presentación *CajaDispatchAction*, la cual empleando una referencia a la clase *CajaServiceManager* accederá al *EJB CajaBean* el cual realizará las operaciones que le han sido solicitadas o bien utilizará otros controladores de acceso a otros EJB's para realizar las operaciones que sean necesarias.

Además, en este diagrama también podemos apreciar como la clase CajaDispatchAction hace uso de las clases *ReportMangerFactory* y *ReportDSFactory* empleadas para realizar un informe o listado que muestre por ejemplo el contenido de un rango determinado de cajas, que nos permita imprimir un rango de etiquetas, etc.

3.5.1.4 Acciones sobre consultas

Este apartado relaciona las clases de presentación encargadas de recibir las solicitudes de consultas de los usuarios sobre diferentes datos (asuntos, cajas, movimientos, intervinientes, etc) que tenemos en el sistema de archivo con las clases de la lógica de negocio encargadas de realizar dichas operaciones de consulta.

En el siguiente diagrama de clases podemos apreciar que todas las operaciones de consulta de la base de datos del sistema de archivo que solicita el usuario son recibidas por la clase de la lógica de presentación *ConsultaDispatchAction*, la cual accederá a la clase encargada de controlar un determinado *EJB* dependiendo de la consulta que se quiera realizar.

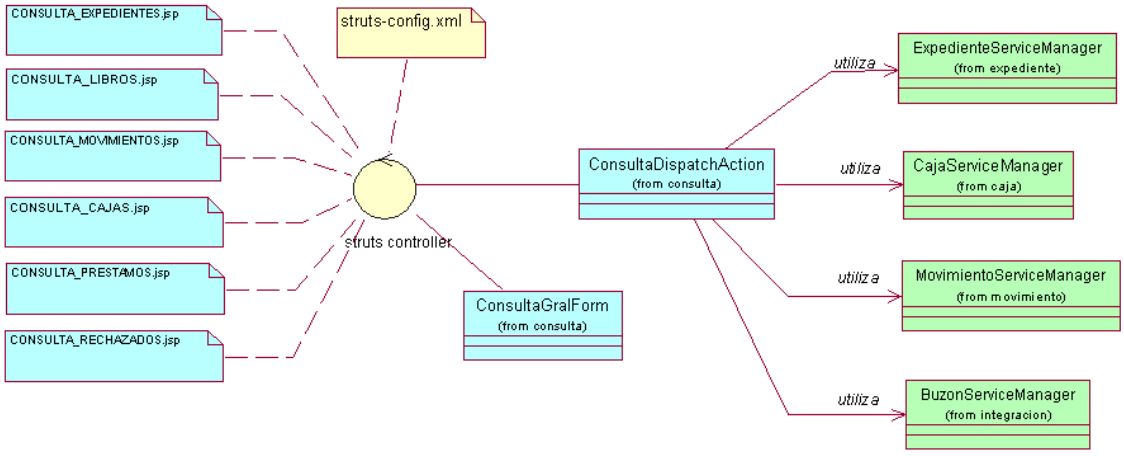

Figura 25. Relación clases consulta lógica presentación con negocio

.

Como se puede apreciar, la clase *ConsultaDispatchAction* tendrá acceso a los controladores,

- **ExpedienteServiceManager**, que nos permitirá tener acceso al *EJB ExpedienteBean* y poder realizar consultas sobre cualquier dato relacionado con los expedientes (asuntos, libros, tomos, etc).

- **CajaServiceManager**, que nos permitirá acceder al *EJB CajaBean* y realizar consultas sobre cualquier dato relacionado con las cajas del sistema de archivo y su contenido.

- **MovimientoServiceManager**, nos permitirá acceder al *EJB MovimientoBean* y realizar consultas sobre los movimientos que se han hecho sobre un determinado expediente del sistema de archivo.

- **BuzonServiceManager**, nos permitirá acceder al *EJB BuzonBean* y realizar consultas sobre las diferentes solicitudes de entrada y respuesta que se han recibido y enviado.

3.5.1.5 Acciones sobre movimientos

Este apartado relaciona las clases de presentación encargadas de recibir las solicitudes de operaciones sobre los movimientos que se van a realizar o se han realizado sobre los expedientes con las clases de la lógica de negocio encargadas de realizar dichas operaciones de movimiento.

En el siguiente diagrama de clases podemos apreciar que todas las operaciones de movimientos sobre los expedientes que solicita el usuario son recibidas por la clase de la lógica de presentación *MovimientoDispatchAction*, la cual utilizará los métodos de la clase *MovimientoServiceManager* para llamar a los métodos que tiene el *EJB MovimientoBean* y realizar las operaciones solicitadas.

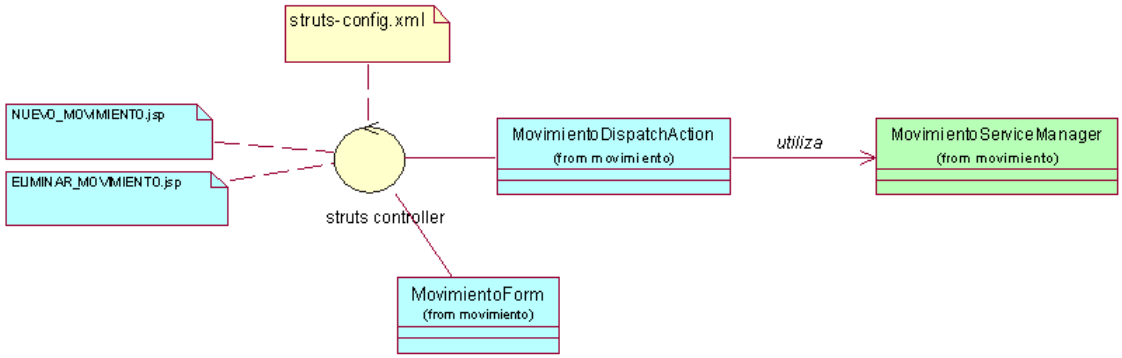

Figura 26. Relación clases movimiento lógica presentación con negocio

3.5.1.6 Acciones sobre integración

Este apartado relaciona las clases de presentación encargadas de generar operaciones sobre los buzones de entrada y salida y las clases de la lógica de negocio encargadas de procesar dichas operaciones que se realizan sobre ambos buzones, es decir de procesar las solicitudes de entrada y de generar las operaciones de salida.

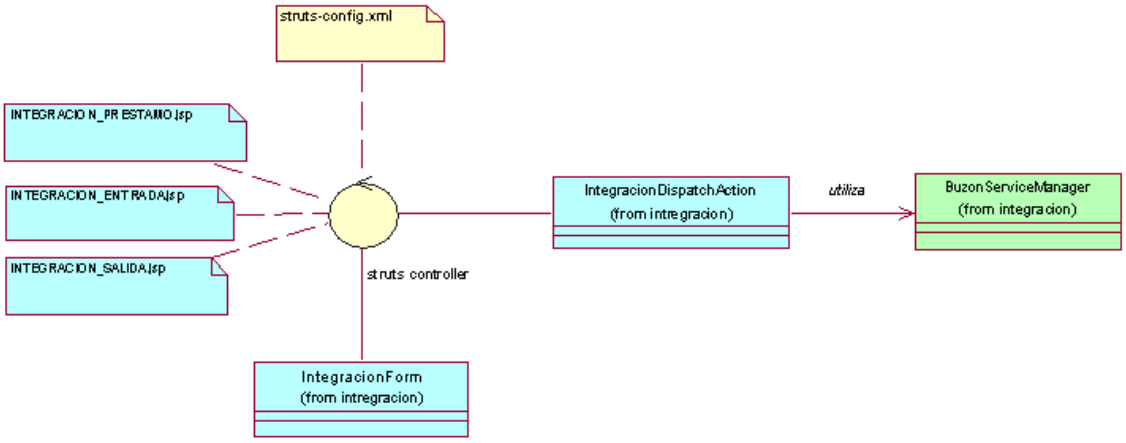

Figura 27. Relación clases buzón lógica presentación con negocio

En el anterior diagrama de clases podemos apreciar que todas las operaciones de integración que solicita el usuario son recibidas por la clase de la lógica de presentación *IntegracionDispatchAction*, la cual utilizará los métodos de la clase *BuzonServiceManager* para acceder al *EJB BuzonBean* y llamar al método correspondiente de este *EJB*.

3.5.1.7 Acciones sobre usario

Este apartado relaciona las clases de presentación encargadas de generar operaciones de administración de usuarios y las clases de la lógica de negocio encargadas de procesar dichas operaciones.

En el siguiente diagrama de clases podemos apreciar que todas las operaciones de administración de usuarios que solicita un usuario habilitado son recibidas por la clase de la lógica de presentación *UsuarioDispatchAction*, la cual utilizará los métodos de la clase *UsuarioServiceManager* para acceder al *EJB UsuarioBean* y llamar al método correspondiente de este *EJB*.

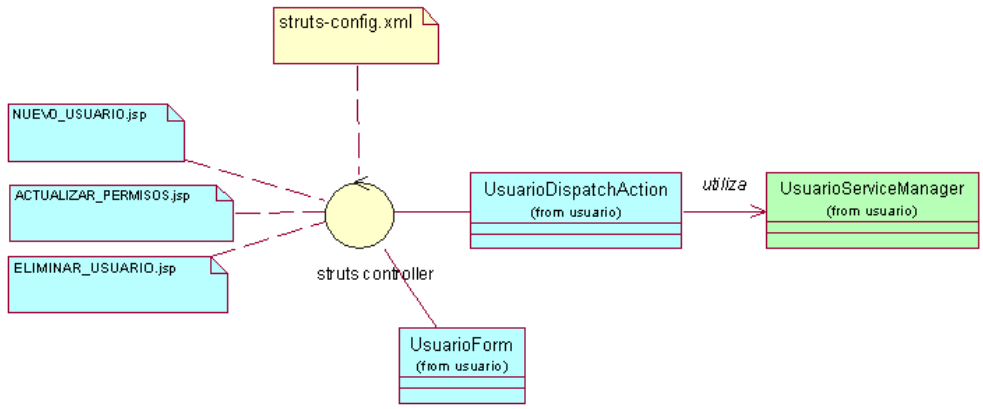

Figura 28. Relación clases usuario lógica presentación con negocio

3.5.2 Relación entre la lógica de negocio y la lógica de datos

Como hemos visto en el punto anteriormente el acceso a la base de datos para obtener información se realiza utilizando objetos que pertenecen a las clases que implementan el patrón "*Data Access Object*". Estas clases emplearán sentencias *SQL*, definidas en el fichero *XML* que tienen asociado, para realizar las correspondientes operaciones sobre la base de datos.

La utilización de estos objetos se realizará desde los *EJB's* de sesión que hemos diseñado. En los siguientes puntos describiremos cuales son las clases "*Data Access Object*" que emplean cada uno de los *EJB's* de sesión.

- **ExpedienteBean**, como se ha especificado anteriormente, es el *EJB* de sesión encargado de realizar todas las operaciones sobre los expedientes. Para realizar estas operaciones necesitará utilizar las clases *AsuntoDAO*, *LibroDAO*, *TomoDAO*, *OrganoDAO*, para poder acceder y manejar toda la información que esté relacionada con un expediente.

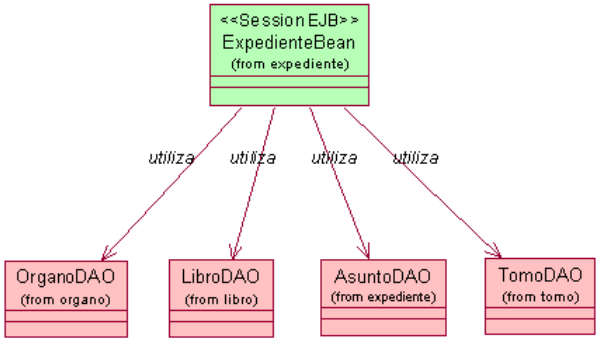

Figura 29. Relación entre Lógica de datos y negocio (ExpedienteBean)

- **MovimientoBean**, es el *EJB* de sesión encargado de realizar todas las operaciones sobre los diferentes movimientos que se realizan sobre los expedientes. Para realizar estas operaciones necesitará utilizar la clase *MovimientoDAO*, para poder acceder y manejar toda la información que este relacionada con un movimiento del expediente.

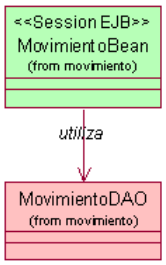

Figura 30. Relación entre Lógica de datos y negocio (MovimientoBean)

- **IntervinienteBean**, es el *EJB* de sesión encargado de realizar todas las operaciones sobre los diferentes intervinientes que están relacionados con un determinado expediente. Para realizar estas operaciones necesitará utilizar la clase *IntervinienteDAO*, para poder acceder y manejar toda la información que este relacionada con los intervinientes de un expediente.

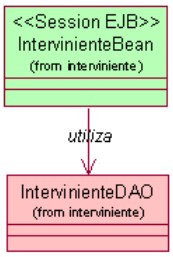

Figura 31. Relación entre Lógica de datos y negocio (IntervinienteBean)

- **CajaBean**, es el *EJB* de sesión encargado de realizar todas las operaciones de mantenimiento de las cajas que se encuentran en el sistema de archivo. Para realizar estas operaciones necesitará utilizar la clase *CajaDAO*, para poder acceder y manejar toda la información que este relacionada con los cajas que se encuentran en el sistema de archivo.

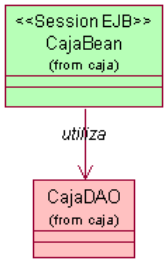

Figura 32. Relación entre Lógica de datos y negocio (CajaBean)

- **ListadoBean**, es el *EJB* de sesión encargado de realizar las operaciones que nos permiten realizar diferentes tipos de listado y cálculos de estadísticas sobre la base de datos. Para realizar estas operaciones necesitará utilizar las clases *ExpedienteDAO*, *LibroDAO*, *TomoDAO*, *IntervinientesDAO*, etc, es decir, este *EJB* de sesión tendrá acceso aquellas clases que implementen el patrón "*Data Access Object*" para poder realizar cualquier listado que se le pida.

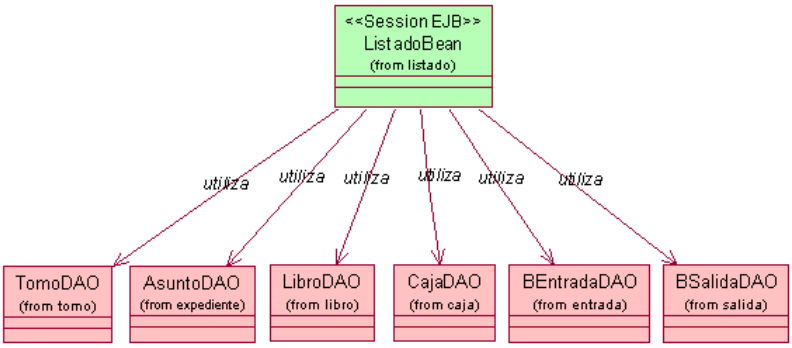

Figura 33. Relación entre Lógica de datos y negocio (ListadoBean)

- **UsuarioBean**, es el *EJB* de sesión encargado de mantener, controlar y manejar, los usuarios que pueden acceder a la aplicación del Sistema de Archivo, así como de los permisos que tiene cada uno de los usuarios y que establecen las operaciones que estos pueden realizar utilizando esta aplicación. Para realizar estas operaciones necesitará utilizar las clases *UsuarioDAO* y *RolDAO* que le permitirán acceder, crear, actualizar y utilizar la información relacionada con los usuarios y los roles que tienen asignados cada uno de los usuarios.

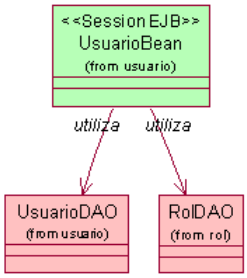

Figura 34. Relación entre Lógica de datos y negocio (UsuarioBean)

- **BuzonBean**, es el *EJB* de sesión encargado de realizar las operaciones que nos permiten controlar la llegada de nuevas solicitudes al sistema de archivo y el envío de operaciones hacia otros órganos que operan con el sistema de archivo. Para realizar estas operaciones necesitará utilizar las clases *BEntradaDAO* y *BSalidaDAO* que le permitirán acceder, crear, actualizar y utilizar la información relacionada con la recepción y envío de solicitudes al sistema de archivo y hacia los diferentes órganos que operan con el sistema de archivo.

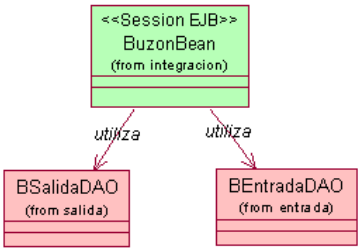

Figura 35. Relación entre Lógica de datos y negocio (BuzonBean)

- **BloqueoBean**, es el *EJB* de sesión encargado de controlar las operaciones de bloqueo y desbloqueo de un expediente que esta siendo editado o ha finalizado su edición por parte de un cliente. Para realizar estas operaciones necesitará utilizar la clase *BloqueoDAO* que le permitirá acceder, crear, actualizar y utilizar la información relacionada con el bloqueo y desbloqueo de expedientes que están siendo editados o ha finalizado su edición.

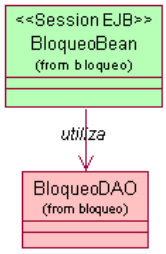

Figura 36. Relación entre Lógica de datos y negocio (BloqueoBean)

- **NumeradorBean**, es el *EJB* de sesión encargado de mantener los identificadores que se emplean como claves internas, para identificar los datos que mantenemos en la base de datos, como los expedientes, cajas, movimientos, etc. Para realizar estas operaciones necesitará utilizar la clase *NumeradorDAO* que le permitirá acceder, crear, actualizar y utilizar los identificadores que se mantienen en la base de datos.

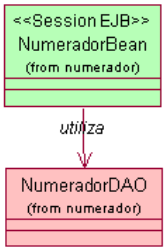

Figura 37. Relación entre Lógica de datos y negocio (NumeradorBean)

Con esto tenemos el diseño completo del sistema y podemos pasar a ver las tecnologías y herramientas que se utilizarán para llevar a cabo la implementación y la puesta en producción.

4 MARCO TECNOLÓGICO

En este capítulo, en primer lugar presentaremos las herramientas empleadas durante el desarrollo del proyecto. Seguidamente veremos las herramientas requeridas para un correcto despliegue de la aplicación en el servidor de producción, y seguidamente las necesarias para una correcta ejecución en los equipos clientes. Por último describiremos brevemente el proceso de despliegue.

4.1 Herramientas empleadas en el desarrollo

Las principales herramientas empleadas en el proceso de desarrollo han sido:

- **WSAD 5.1** (Websphere Studio Application Developer)
  Es un entorno de desarrollo de IBM basado en Eclipse, y al que añade una serie de facilidades para el desarrollo de aplicaciones J2EE, mediante los plugings integrados

- **Toad** for Oracle v8.6
  Toad [2] es una aplicación de software de desarrollo SQL y administración de base de datos, considerada una herramienta útil para los Oracle DBAs (administradores de base de datos). TOAD está ahora disponible para las siguientes bases de datos: Oracle Database,Microsoft SQL Server, IBM DB2, y MySQL.

- **Notepad++**
  Notepad++ [2] es un editor de código fuente libre, que admite varios lenguajes de programación y se ejecuta en Microsoft Windows. Hace tiempo ya puede utilizarse en GNU/Linux mediante Wine, y reciente-mente ha añadido Drag and drop (arrastrar y soltar) de archivos para este Sistema Operativo libre.

Admite autocompletado, búsqueda y reemplazo usando expresiones regulares, edición con pantalla dividida, Zoom de texto, marcadores de texto (bookmark), y resaltado de paréntesis. Admite macros y plugins

Este proyecto, basado en el componente de edición Scintilla, está escrito en C++ utilizando directamente la API de Win32 y STL, lo que asegura una velocidad mayor de ejecución y un tamaño más reducido del programa final. Se distribuye bajo los términos de la Licencia Pública General de GNU.

- **SmartCVS**

  El Concurrent Versions System (CVS), también conocido como Concurrent Versioning System , [2] es una aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros (código fuente principalmente) que forman un proyecto (de programa) y permite que distintos desarrolladores (potencialmente situados a gran distancia) colaboren.

  CVS utiliza una arquitectura cliente-servidor: un servidor guarda la(s) versión(es) actual(es) del proyecto y su historial. Los clientes se conectan al servidor para extraer una copia completa del proyecto.

  Los clientes pueden también comparar diferentes versiones de archivos, solicitar una historia completa de los cambios, o obtener una "foto" histórica del proyecto tal como se encontraba en una fecha determinada o en un número de revisión determinado. Muchos proyectos de código abierto permiten el "acceso de lectura anónimo", significando que los clientes pueden sacar y comparar versiones sin necesidad de teclear una contraseña; solamente el ingreso de cambios requiere una contraseña en estos casos.

  Los clientes también pueden utilizar la orden de actualización con el fin de tener sus copias al día con la última versión que se encuentra en el servidor. Esto elimina la necesidad de repetir las descargas del proyecto completo.

En nuestro caso hemos utilizado la aplicación SmartCVS de la empresa Syntevo Software Devolopment.

4.2 Tecnologías empleadas durante el despliegue

A continuación se reproduce un esquema con los principales elementos tecnológicos implicados en el despliegue de la solución.

4.2.1 Servidor de aplicaciones J2EE

En él se instalará el Servidor JBoss y se desplegará la aplicación de Archivo de Gestión.

JBoss AS [2] es el primer servidor de aplicaciones de código abierto, preparado para la producción y certificado J2EE 1.4, disponible en el mercado, ofreciendo una plataforma de alto rendimiento para aplicaciones de e-business. Combinando una arquitectura orientada a servicios revolucionaria con una licencia de código abierto, JBoss AS puede ser descargado, utilizado, incrustado, y distribuido sin restricciones por la licencia. Por este motivo es la plataforma más popular de middleware para desarrolladores, vendedores independientes de software y, también, para grandes empresas.

Las características destacadas de JBoss incluyen:

- Producto de licencia de código abierto sin coste adicional.
- Cumple los estándares.
- Confiable a nivel de empresa
- Incrustable, orientado a arquitectura de servicios.
- Flexibilidad consistente
- Servicios del middleware para cualquier objeto de Java
- Ayuda profesional 24x7 de la fuente

Para el despliegue, el servidor empleado será la versión 3.2.7 de JBoss que soportará la versión 1.3 de J2EE.

El dimensionamiento de este servidor de aplicaciones tendrá las siguientes características mínimas recomendadas para el despliegue de la aplicación de Archivo de Gestión:

- Procesador de frecuencia igual o superior a 2 GHz
- RAM >= 1 Gb (Preferiblemente 2 Gb)
- Disco Duro: 400 Mb libres

Estas características mínimas recomendadas pueden variar ligeramente de acuerdo a los requerimientos mínimos del sistema operativo soportado por JBoss (Windows 2000, Windows XP, Windows 2003, Solaris, distribución de Linux) que se vaya a utilizar.

En cualquier caso, se considera que éste servidor no va a soportar el Gestor de Base de Datos, residiendo éste en otro equipo servidor.

En las características finales debería considerarse también la posibilidad de crecimiento y escalabilidad de cara el despliegue futuro de otras posibles aplicaciones J2EE.

4.2.2 Servidor de Base de Datos

Como Gestor de Base de Datos, la aplicación está certificada para versiones de Oracle 8i y superiores.

Oracle [2] es un sistema de gestión de base de datos relacional (o RDBMS por el acrónimo en inglés de Relational Data Base Management System), fabricado por Oracle Corporation.

Se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando su:

- Soporte de transacciones.
- Estabilidad.
- Escalabilidad.
- Soporte multiplataforma.

4.2.3 Requisitos del equipo cliente

El cliente precisará del siguiente software de base externo a la aplicación:

- Navegador de documentos HTML con soporte para Javascript.
- Visualizador de documentos PDF

Como se puede apreciar los requisitos de los clientes son muy básicos y de carácter gratuito.

4.3 Guía de despliegue

La aplicación que se suministra consta únicamente de dos archivos:

- Archivo.ear → aplicación propiamente dicha
- oracle-ds.xml→ descriptor de despliegue de JBoss.

El fichero oracle-ds.xml es un fichero de despliegue propio de Jboss donde se configura la conexión a la base de datos.

Un ejemplo del fichero es el siguiente. Se han marcado en negrita los parámetros principales:

- Nombre del enlace JNDI en el cual se registrará el datasource (la aplicación espera que se registre bajo el nombre "jdbc/OracleDS")

- String de la conexión a base de datos.

- Usuario y password de la base de datos.

- Dimensionamiento del pool de conexiones (mínimo número de conexiones a abrir y máximo): Al margen de estas conexiones, la aplicación abre otro pool de 10 conexiones extra para uso interno: recuperación de códigos.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ===================================================================== -->

<!-- DataDirect Data Sources -->

<!-- ===================================================================== -->

<!--

See the generic_ds.xml file in the doc/examples/jca folder

for examples of properties and other tags you can specify

in data sources

-->

<datasources>

<local-tx-datasource>

<jndi-name>jdbc/OracleDS</jndi-name>

<connection-url>jdbc:oracle:thin:@192.168.60.81:1521:miro</connection-url>

<driver-class>oracle.jdbc.driver.OracleDriver</driver-class>

<user-name>aexp</user-name>

<password>****</password>

<!--pooling parameters-->

<min-pool-size>5</min-pool-size>

<max-pool-size>10</max-pool-size>

<blocking-timeout-millis>5000</blocking-timeout-millis>

<idle-timeout-minutes>15</idle-timeout-minutes>

</local-tx-datasource>

</datasources>
```

El despliegue de la aplicación consta de los siguientes pasos:

1. Establecer, si no existe previamente, la variable de entorno JAVA_HOME al directorio de instalación de una JDK válida. (Recomendable una versión 1.4.x o superior).

2. Lanzar el Jboss en modo servidor DEFAULT o en el modo que se estime adecuado: ..\Bin\run.bat (como parámetro se le puede pasar el servidor, si no recibe ningun parámetro abre el default).

3. Copiar librería de Oracle en la carpeta $JBOSS/lib.

4. Configurar un archivo de datasource (oracle-ds.xml) y depositarlo en el directorio de deployment (server\...\deploy)

5. Desplegar la aplicación copiando el EAR correspondiente al directorio de deployment.

5 PROBLEMAS DETECTADOS

En este punto se pretende poner de manifiesto todos los problemas que han ido surgiendo a lo largo del desarrollo del proyecto. Además, en el siguiente punto se expondrán algunas soluciones alternativas, que darían soluciones adecuadas a estos problemas.

Los problemas encontrados han sido los siguientes:

- Documentación desactualizada y poco fiable. Durante el proceso de desarrollo de software no siempre se le da la importancia que merece a la documentación. Por lo que es fácil que esta no sea del todo completa. También es posible que no esté actualizada con los últimos cambios o evoluciones que el software puede haber sufrido, incluso después de la entrega del proyecto

- Un proyecto de migración como este, es un proyecto complejo. Esta complejidad es debida a que el desarrollador tiene que tener un profundo conocimiento de ambos frameworks, tanto el de origen como el destino. A esto se suma la necesidad de capturar todo el modelo de negocio de la aplicación de origen en el nuevo producto. Esto no siempre es fácil ya que, para ciertas partes de este modelo, no se tiene más documentación que el propio código.

- La utilización del nuevo framework, y la obligación de utilizar las clases que este nos proporciona, ha obligado a reescribir gran parte del código de la aplicación. Esto obliga a rehacer un trabajo ya hecho por necesidades puramente tecnológicas, cuando en realidad el modelo de negocio es idéntico.

- Se ha sustituido código muy fiable, ya que ha sido utilizado en un entorno de producción durante bastante tiempo, por otro nuevo. Este código es de menor fiabilidad, ya que únicamente ha sido sometido a pruebas durante el desarrollo en un entorno de preproducción, sin usuarios finales.

- Al ser un proyecto de migración, se han infravalorado los recursos necesarios para el desarrollo, lo que ha llevado a un incumplimiento de los plazos previstos inicialmente. Además estos problemas hacen que la rentabilidad del proyecto se vea reducida.

- En la nueva aplicación quedan "restos" de código antiguo que no se utilizan. Por ejemplo métodos para la gestión de usuarios que han sido sustituidos por los proporcionados para la plataforma. Este código residual no siempre es fácil de detectar, y dedicar mucho tiempo a realizar una buena limpieza, reduciría más aún la rentabilidad del proyecto. Todo esto reduce la calidad del código de la aplicación.

- Detección tardía de errores en la captura de requisitos, lo que ocasiona que se tenga que deshacer trabajo ya realizado.

- Si surge la necesidad de migrar alguna otra aplicación a este mismo framework, no solo volverían a surgir estos mismos problemas, sino que el trabajo realizado en este proyecto sería difícilmente reutilizable. El único bagaje que aportaría sería la experiencia de los desarrolladores en las diferentes plataformas (siempre que sean los mismos).

- Que una aplicación funcione en el servidor de pruebas incluido en Websphere, no garantiza un correcto funcionamiento una vez desplegado en JBOSS. De hecho es común que no sea así JBOSS es mucho más sensible a algunos errores que en Websphere.

Estos son los problemas que he creído más relevantes de los que he podido encontrar durante el desarrollo en el siguiente punto, plantearé una posible solución a la mayoría de ellos.

6 MDA COMO SOLUCIÓN ALTERNATIVA

Una vez vistos los problemas que han ido surgiendo a lo largo del proyecto, vamos a presentar algunas soluciones alternativas, incluidas en los contenidos del máster, que podrían ser soluciones más adecuadas a estos problemas.

Uno de los temas que más se han tratado en los contenidos del máster es la arquitectura MDA. A continuación, después de realizar una breve introducción de esta arquitectura, veremos cómo su utilización habría evitado la mayoría de los problemas que comentamos en el punto anterior. No se profundizará en esta arquitectura ya que esto se realiza en la documentación de la bibliografía, y únicamente se pretende que sirva de apoyo para entender sus beneficios.

6.1 Presentación Model Driven Architecture

La arquitectura dirigida [1] por modelos es un framework establecido por la OMG (Object Managment Group) para los nuevos desarrollos que sigan un nuevo paradigma de programación, que no es otro que MDE,(Model Driven engineering).

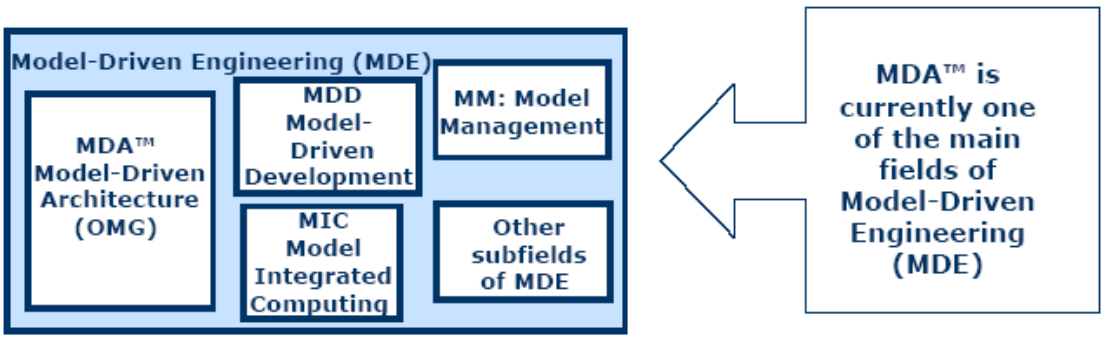

Figura 38. Model-Driven Engineering

En este nuevo paradigma los modelos toman el papel protagonista en el proceso desarrollo de software, ya que mediante una serie de transformaciones nos permitirán obtener el código de la aplicación final. Esto es posible gracias a que los modelos son formales, lo que permite que sean computables.

La utilización de esta arquitectura, permite obtener beneficios importantes en aspectos fundamentales, como son la productividad, la portabilidad, la interoperabilidad, el mantenimiento y la calidad del producto software producido.

La arquitectura MDA se compone de tres tipos de modelos y se basa en las transformaciones entre ellos.

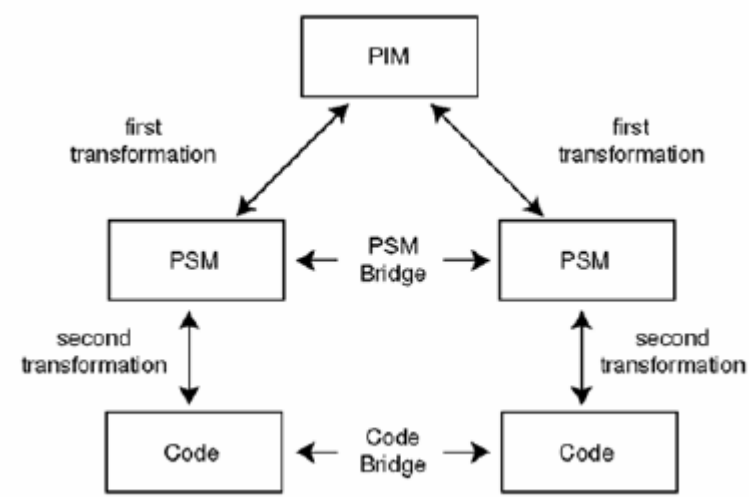

Figura 39. Transformaciones entre los modelos

Estos tres tipos de modelos son:

PIM (Platform Independent Model)

Este es un modelo con un alto nivel de abstracción, totalmente independiente de la tecnología utilizada en la implementación. Este tipo de modelos tiene la ventaja de que puede ser producido y supervisado por los expertos en el dominio del problema, olvidándose de los detalles más tecnológicos.

PSM (Platform Specific Model)

El PIM puede ser transformado en uno o más PSM que son modelos que permiten especificar el sistema para su implementación en una plataforma específica. De este modo, vemos como a partir de un mismo PIM, podemos obtener aplicaciones multiplataforma, y facilitar su migración a tecnologías futuras permitiendo su reutilización.

Para realizar estas transformaciones, el estándar propuesto por la OMG es el lenguaje de transformación de modelos QVT (Query View Transformations). Aunque existen otras soluciones como son XSL y AWK.

CM (Code Model)

Es el último paso la transformación de cada PSM a código. Debido a que el PSM se adapta a cada tecnología, la transformación a código se pude realizar de forma automática.

Para almacenar esta información la OMG propone como estándar XMI (XML Metadata Interchange) cuyo propósito es facilitar el intercambio entre herramientas de modelado basadas en UML.

6.2 Revisión de los problemas

Ahora vamos a hacer un repaso a los problemas que presentamos en el punto anterior y veremos como MDA actuaría sobre ellos.

- *Documentación desactualizada y poco fiable*. Con MDA, los modelos de diseño y análisis no se abandonan y se mantienen siempre actualizados. Hay trazabilidad desde la especificación hasta la implementación.

- *Proyecto complejo, el desarrollador tiene que conocer las plataformas de origen y destino*. Gracias a MDA únicamente tiene que comprender su modelo PIM, y preocuparse de hacer u obtener un PSM adecuado para su plataforma.

- *A veces hay que recurrir al código para algunos detalles del modelo de negocio*. No en MDA, el modelo de negocio debe estar completamente capturado en el PIM.

- *Cuando una aplicación va dirigida a un cliente tan especializado como es el estamento judicial, con una jerga propia no siempre es fácil entenderse.*

  Por un lado, es difícil para el técnico capturar todos los requerimientos de su cliente. Por otro, para el cliente no es posible detectar problemas en la captura de los requisitos, si se utilizan otras técnicas más convencionales como los diagramas entidad-relación. Por esto se detectan errores en fases avanzadas del desarrollo, en las que ya se han invertido unos recursos, en algo, que puede no ser lo que el cliente necesita.

  En definitiva mejora la comunicación con el cliente, y permite involucrarlo en el proceso de desarrollo. Esto es debido a que el modelo PIM permite acercar y mejorar la comunicación con los clientes, al ser más fácil de entender por estos.

- *Por necesidades tecnológicas hay que reescribir código que forma parte del modelo de negocio lo que puede contribuir a introducir errores en este*. En MDA el PIM permanece inalterado, por lo que el modelo de negocio se conserva intacto.

- *Se reduce la calidad del código, debido a que quedan restos residuales de la aplicación anterior y es fácil que se introduzca algún error accidentalmente.* Con la generación de código nos olvidamos del código residual, y si algo falla, fallará siempre por lo que será más fácil de detectar y solucionar.

- *Si hay que realizar la migración de otra aplicación el trabajo realizado es difícilmente reutilizable.* Con MDA la migración será mucho más sencilla, ya que podremos aprovechar las transformaciones del proyecto anterior, y la transformación de PSM a código, será prácticamente automática.

- *Incumplimiento de los plazos previstos y detección tardía de errores en la captura de requisitos.* Estos problemas tienen, en parte, su origen en la utilización de un ciclo de vida en cascada.

  Para evitar este tipo de problemas, en los contenidos del máster, también se presentan metodologías ágiles que mejoran la captura de estos requisitos. Un ejemplo es RUP, que involucra a los clientes en el proceso de desarrollo con entregas continuas de software utilizable, que incrementalmente van conformando la versión final.

  En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados.

- *Que una aplicación funcione en el servidor de pruebas incluido en Websphere, no garantiza un correcto funcionamiento una vez desplegado en JBOSS.* Esto pone de manifiesto la necesidad de realizar pruebas en JBOSS de cada nueva versión antes de darla por cerrada. Para depurar los posibles errores JBoss es de poca ayuda, por lo que es interesante la opción que nos proporciona Websphere, de depurar utilizando un servidor externo.

Aquí acaban las posibles soluciones a los problemas expuestos en el punto anterior, con lo que pasamos a ver las conclusiones finales.

7 CONCLUSIONES

A lo largo de este documento se ha detallado un desarrollo práctico, relacionado con la actividad profesional propia de un titulado en informática. Concretamente dentro del campo de la Ingeniería del Software y los Sistemas de Información. Ambas ramas junto a la de métodos formales dan título al grado de máster que se pretende alcanzar con este trabajo.

Para alcanzar este objetivo principal, hemos comenzado contextualizando el desarrollo para una empresa y un cliente en concreto, y justificando el desarrollo en base a unas necesidades de este cliente.

En el núcleo central del trabajo se han presentado los requisitos de la aplicación, y un diseño en base a estos.

Una vez finalizado el desarrollo de la aplicación, se han recogido los problemas que han aparecido, y se ha intentado, de alguna manera, presentar soluciones a estos, en las que intervienen los contenidos del máster.

Este trabajo, pretende ser un ejemplo de este tipo de desarrollos, y sirve para poner de manifiesto las ventajas que aportaría la utilización de este tipo de técnicas. Por lo que espero que, además de reafirmarme personalmente, en la convicción de que este cambio de paradigma sería muy beneficioso para la industria, sirva de motivación para otros profesionales en su utilización.

No obstante para cambiar la visión de las empresas y sus dirigentes, que durante años se han centrado en la producción rápida de código, dejando a los modelos en un segundo plano, queda un largo camino.

Para ello es fundamental el papel de las universidades y los grupos de investigación, en su papel formador e informador y la aparición de herramientas cada vez mejores que faciliten esta transición.

8 BIBLIOGRAFÍA

[1] Insfrán, D. E. (2007). Doctoral Seminar Model-Driven Architecture.

[2] Wikipedia

[3] Jim Keogh. Manual de referencia J2EE(MC Graw Hill)

[4] Guía del IEEE para la Especificación de Requisitos Software (IEEE std.830-1993). Departamento Sistemas Informáticos y Computación U.P.V

[5] Real Decreto de Modernización de los Archivos Judiciales.

9 ANEXO

A continuación se incluye como anexo el Manual de Usuario de la aplicación de Archivo Judicial. El documento puede resultar interesante, ya que permite conocer en profundidad la aplicación.

En este documento, se pueden encontrar de manera bastante detallada. Los diferentes procesos que pueden realizar los usuarios y una breve explicación de cómo realizarlos correctamente.